Experiences and Lessons from Introducing Model-Based Analysis in Brown-Field Product Family Development

Jacques Verriet[1], Bram van der Sanden[1], Gijs van der Veen[2], André van Splunter[2], Sam Lousberg[3], Martijn Hendriks[4] and Twan Basten[4,1]

[1] *TNO-ESI, High Tech Campus 25, Eindhoven, Netherlands*

[2] *ITEC, Jonkerbosplein 52, Nijmegen, Netherlands*

[3] *Nobleo, Heggeranklaan 1, Eindhoven, Netherlands*

[4] *Electronic Systems Group, Eindhoven University of Technology, De Zaale, Eindhoven, Netherlands*

Keywords: Product Family, Productivity Modeling, Model Validation, Abstraction, Brown-Field, Lessons Learned.

Abstract: Product family development facilitates reuse across all phases of systems engineering; in case of model-based systems engineering, this reuse involves the models as well. Introducing a model-based way of working is challenging, especially for product family development. This paper describes a case of introducing a model-based way of working in brown-field product family development. We explain how we developed a *master model*, i.e. a library of model elements, to predict and optimize the productivity of a family of industrial production systems. Using this master model, we construct models of existing and yet-to-be-developed product family members by configuring and combining the appropriate library elements. We use system and model execution traces to validate the productivity models. For this, we developed a *master transformation*, i.e. a library of execution trace transformation rules, to unify system and model execution traces. Besides the master model and the master transformation, we present lessons learned regarding the introducing a model-based way of working. This proves both technically and organizationally complex, especially for brown-field product family development, but besides the intended prediction and optimization, it brings benefits with respect to capturing domain knowledge and system validation.

1 INTRODUCTION

Many companies do not develop a single system, but a family of similar systems. The development of such a family facilitates reuse throughout all phases of systems engineering; this reuse lowers costs, decreases development times, and increases system quality (van der Linden et al., 2007).

In this paper, we focus on the logistics of a family of high-performance production systems. When extending the product family with a new variant, new technology has to be introduced. To introduce a family member with a higher productivity, i.e. a higher number of products produced per hour, one could use faster components. To support the manufacturing of different products, one needs new types of components. To support such changes, one has to redesign the corresponding production logistics.

To reduce development cost and effort, one would like to assess the feasibility of a yet-to-be-developed system variant as early during development as possible. Development and integration of new hardware and software is both time consuming and costly. Instead, one can choose a model-based approach to assess variant feasibility. A model-based approach allows creating accurate models of existing system variants to predict the feasibility of yet-to-be-developed variants. Papparapurath et al. (Parappurath et al., 2013) refer to this as *predicting the past* and *exploring the future*.

Contribution. In this paper, we introduce a model-based way of working to the development of an existing family of production systems. Introducing a model-based way of working in a brown-field situation is not straightforward, as the development processes, that evolved over time, typically do not facilitate this. We develop a so-called *master model*, i.e. a library of model elements, from which we derive system variant models. Using this master model, we predict the productivity of existing and yet-to-be-developed system variants.

Verriet, J., van der Sanden, B., van der Veen, G., van Splunter, A., Lousberg, S., Hendriks, M. and Basten, T.
Experiences and Lessons from Introducing Model-Based Analysis in Brown-Field Product Family Development.
DOI: 10.5220/0011785900003402
In *Proceedings of the 11th International Conference on Model-Based Software and Systems Engineering (MODELSWARD 2023)*, pages 226-236
ISBN: 978-989-758-633-0; ISSN: 2184-4348

Before we can use a model to accurately predict system productivity, we need to validate the model's correctness. This model validation is especially challenging in a brown-field situation as the information needed to validate newly developed models is typically not readily available. To support the validation of our productivity models, we develop a *master transformation*, i.e. a library of transformation rules. These rules bridge the gap between the system variants and the productivity models by transforming execution traces originating from both system variants and models into execution traces in a unified form, which allows validation by comparison.

Besides the master model and the master transformation, we share our experiences of introducing a model-based way of working in brown-field product family development by formulating generic lessons learned, using the specific family of production systems as an illustration. Our results show that introducing a model-based way of working in brown-field product family development is more complex than for single system development: the challenges are sometimes purely technical, but typically (also) organizational. Besides the benefit of productivity prediction, our use case demonstrates benefits of a model-based way of working with respect to capturing system knowledge and system validation.

Outline. This paper is organized as follows. Section 2 presents related work. Section 3 describes the product family that we used as a use case for introduction of productivity modeling in brown-field development. The model development process is described in Sections 4 and 5; Section 4 addresses the modeling of one system variant and Section 5 the generalization to the entire product family. We present the lessons learned from our modeling experiences in Section 6. Conclusions are presented in Section 7.

2 RELATED WORK

In this paper, we explain the development of a master model for a family of industrial production systems and a corresponding master transformation that unifies execution logs originating from system configurations and from the variant models derived from the master model.

Related is the work of Verriet et al., who develop a predictive model for an existing family of wide-format production printers (Verriet et al., 2018). To predict the timing of the image processing of all system variants, they extract a parameterized performance model from the printers' source code using static analysis and they calibrate this model using regression. Their approach is limited to prediction of the duration of computational actions. Tawhid and Petriu also consider software performance prediction: they present a method to automatically generate Layered Queuing Network (LQN) models from a UML+MARTE specification (Tawhid and Petriu, 2008). These LQN models are used to assess the performance of a software product line with structural and behavioral variation. Our focus is broader than theirs: our master modeling approach addresses both mechanical and software variation.

We use the term *master model* for the parameterized productivity models used to analyze a family of production systems. This term originates from the Computer Aided Design (CAD) domain. In that domain, it involves having a central model repository from which other models are (semi-automatically) derived. Hoffman and Joan-Arinyo (Hoffman and Joan-Arinyo, 1998) present a client-server framework for product master modeling. Clients' views of the central design model automatically get updated after a change in the master model. Sandberg et al. present an application of master modeling for jet engine design (Sandberg et al., 2011).

As our master model is not meant to represent a single system configuration, but a family, our master model has commonalities with the 150% models used in the automotive industry (Grönniger et al., 2008). A 150% model of a family of system configurations describes all possible features of a configuration. Variant models, also called 100% models, are derived from the parameterized 150% model by instantiation. The different variants are typically described using feature models (Kang et al., 1990; Czarnecki and Eisenecker, 2000). Both the master models used in the CAD domain and the 150% models from the automotive domain are descriptive models; they do not allow performance predictions. They could however be used to generate predictive models from.

To be able to validate variant models derived from our master model, we relate the execution of a system variant to the execution of the corresponding model derived from the master model. For this, we use a domain-specific language (DSL) to transform execution traces, which could come from both the system variants and the models. Our master transformation, which is an instance of this DSL, realizes the abstraction needed to bridge this gap by transforming both system and model execution traces into a unified form. Related is the DSL of Gad (Gad, 2017); this DSL is used for importing packet capture data into Trace Compass,[1] but it does not support subsequent transformations.

[1] https://www.eclipse.org/tracecompass/

Our master transformation transforms low-level signal and event data in system execution logs into higher-level action information. Feng et al. and Reiss present other examples of abstraction to bridge the gap between system and model (Feng et al., 2018; Reiss, 2005). To allow inspection of execution traces originating from software systems, they present an approach to identify known patterns to create a view of the trace. In Feng et al.'s work, this involves multiple hierarchical levels (Feng et al., 2018). The work of both Feng et al. and Reiss specifically focuses on software systems; our master transformation does not have this restriction and can be applied to execution traces from any system or model source. Feng et al.'s abstraction considering multiple levels can be achieved by applying the master transformation's DSL multiple times.

3 USE CASE

This paper describes how we introduced productivity modeling for an existing family of industrial production systems. As a use case, we have used ITEC's ADAT3-XF die bonder platform.[2] The die bonders are used for low-cost, high-volume electronic assembly. Figure 1 shows an ADAT3-XF die bonder. The die bonders of the ADAT3-XF platform pick up dies from a diced wafer on a wafer table and attach them to a lead frame held by a product holder. The die bonders' typical use involves long batches of the same product being handled. The goal is to achieve both a high productivity and high product quality. As quality inspections take time, one has to find the best trade-off between these two system qualities.

Figure 1: ADAT3-XF die bonder.

Figure 2 illustrates the overall die bonder process. On its journey, a die gets transported to different processing stations. At the first processing station, the die is picked up. At the second station, it gets inspected for quality; these quality inspections may involve visual inspections for damage or testing of the die. There are optional processing stations, at which a die gets flipped and subsequently inspected again. At the last processing station, the die gets attached to a lead frame. Between the operations on the die, there are index steps which transport the die to the next processing station.

To obtain the desired high productivity, the operations at the ADAT3-XF's processing stations run in parallel. Figure 2 shows that five dies are processed in parallel at the five stations (Pick-up, Inspect 1, Flip, Inspect 2, Attach). To allow this kind of pipelined parallelism, die operations and die transfers need to be synchronized. Similarly, the indexing steps of the wafer table between die pick-ups and those of the product holder between die attachments need to be synchronized with the die transfer and die operations.

Our goal is to quickly and accurately predict the productivity of the different variants of the ADAT3-XF die bonder family. This is challenging because of the system variability. The variability in the die bonder product family can be described in terms of Product, Process and Resource (Meixner et al., 2019).
Product. A source of product variability is the characteristics of the dies being processed, e.g. their dimensions. Another source of product variability is the lead frame to which the dies are attached; this includes the lead frame's substrates (i.e. reel, strip, panel, web).
Process. The lead frame defines the corresponding bonding technique (glue, eutect), and the places where to attach dies. Both influence the process realized by a die bonder, i.e. the steps needed to attach the dies to the lead frame. Another source of process variability is the pipeline (see Figure 2), e.g. the number and type of quality inspections performed and whether dies need to be flipped.
Resource. The lead frame also influences resource variability, because each substrate has its own product holder. Besides the product holder, resource variability includes components with different timing characteristics, e.g. components running at different speeds: the ADAT3-XF family includes variants implementing the same process, but at different speeds.

4 SINGLE SYSTEM MODELING

As a stepping stone for introducing productivity modeling for the family of die bonders, we modeled one variant. A die bonder is a mechatronic systems with a repetitive steady-state behavior. To predict the variant's productivity, we created a productivity model of

[2] https://www.itecequipment.com/products

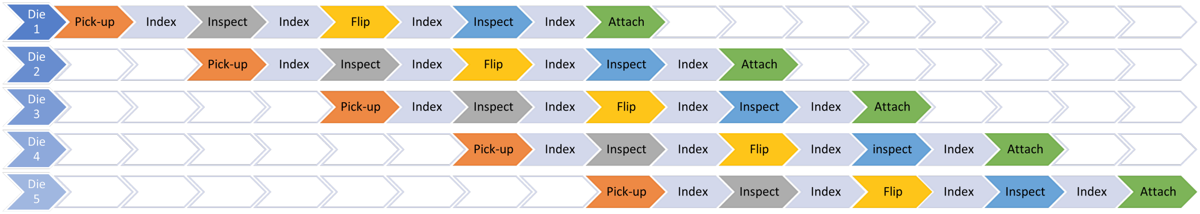

Figure 2: ADAT3-XF die bonder process pipeline.

its steady-state behavior. We distinguish the creation of the model (see Section 4.1) and the validation of the model's correctness (see Section 4.2). As creating an accurate model is not a first-time-right process, we needed to perform both steps several times.

4.1 Model Creation

LSAT[3] is a tool to specify and analyze the logistics of cyber-physical production systems (van der Sanden et al., 2021). LSAT is especially meant for modeling the deterministic logistics of production systems like ITEC's die bonders. Using LSAT, one specifies a system in terms of a platform and an application (see Figure 3). The platform describes the system's resources and the functionality they provide; the application describes the system's behavior by combining resource functionality.

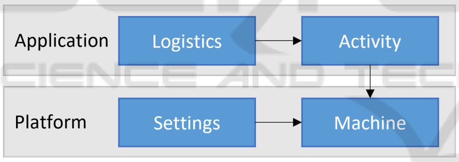

Figure 3: LSAT model structure (adapted from (van der Sanden et al., 2021)).

Platform: To model the platform, we need to specify a machine model and a settings model. LSAT's *machine models* specify the resources with their peripherals, and the actions and movements that these peripherals can perform. The resources mainly involve the components needed to realize steps of a desired process, but they also include synchronization resources. Synchronization resources are used e.g. to avoid collisions between components that move in the same space. Figure 4 shows the resources and the peripherals of the model of a simplified ADAT3-XF system. The main resources are the wafer table, the pushup unit, the die transfer, two Z-motors, and the product holder. In addition, there are two synchronization resources to model dependencies between the operations performed during the journey of a die. Also part of the machine model, but not shown in Figure 4, are the actions and movements that the resources' peripherals can perform.

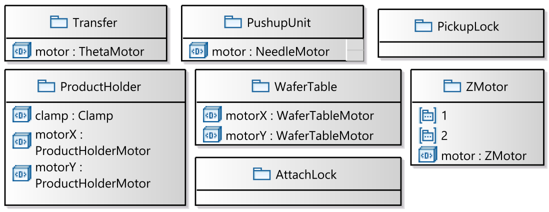

Figure 4: ADAT3-XF machine model.

LSAT's *settings models* capture the physical characteristics of the machine, including the physical coordinates and motion profiles describing motor movements. Settings model specify the characteristics of the machine model's peripherals; Figure 5 shows part of the simplified ADAT3-XF system's settings model. It contains durations for peripheral actions as well as motion parameters and physical locations for motor movements.

```
ZMotor.motor {
  Timings {
    placement = 0.008
  }
  Axis Z {
    Profiles {
      extend (V = 0.7, A = 250, J = 38000, S = 0)
      retract (V = 0.08, A = 250, J = 83333, S = 0)
    }
    Positions {
      neutral = 0.0000
      extended = 0.0012
      retracted = 0.0007
    }
  }
}
```

Figure 5: ADAT3-XF settings model.

Application. On its journey, a die gets picked up from a diced wafer on the wafer table, it visits several processing stations for (quality inspection) process steps, and it gets attached to a lead frame on the product holder. The transfer of the dies between process locations is called indexing. Between two consecutive pick-up steps, the wafer table also makes an indexing step to allow the next die to be picked up. Similarly, the product holder makes indexing steps between attach operations. To describe a system's application, these operations and the transportation in between are

[3] https://www.eclipse.org/lsat/

described as activities in LSAT's *activity models*. Activities are directed acyclic graphs consisting of peripheral actions as defined in the machine model.

LSAT's activities describe the operations performed. An example is the pick-up activity shown in Figure 6. This activity involves two active resources, i.e. ZMotor and PushupUnit, and two resources that are used for synchronization, i.e. PickupLock and WaferTable. The activity diagram shows that the Z-motor is extended, after which the needle extends and pushes a die on the Z-motor's collet. After this die transfer, the Z-motor and the needle retract simultaneously. The synchronization resources get released during the retraction of the Z-motor and the needle. At this point in the retraction, the die transfer and the wafer table can safely start their index step. This parallelism increases the system's productivity.

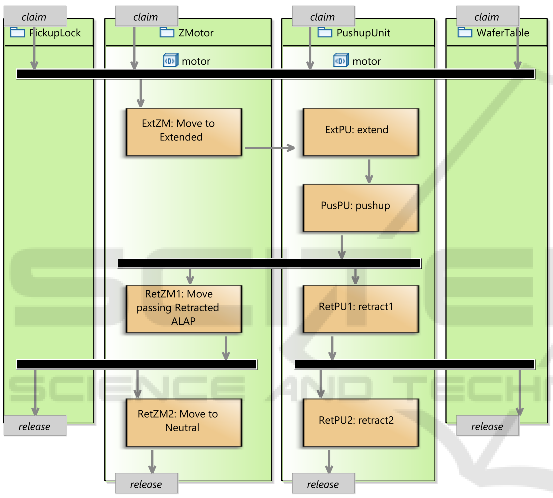

Figure 6: ADAT3-XF pick-up activity.

LSAT's *logistics models* combine activities into a desired overall system process. The dispatching sequence shown in Figure 7 describes a four cycles of the simplified ADAT3-XF configuration. Each cycle involves indexing of the die transfer, the wafer table and the product holder, followed by pick-up and attach activities. Note the the pick-up and attach activities are parameterized; their parameters indicate which Z-motor is used.

The activities in a dispatching sequence interact via the resources that they share (van der Sanden et al., 2021): if an activity shares a resource with a preceding activity, the activity's actions involving this shared resource can only start after the preceding activity has released the resource. If activities do not share any resources, they can run in parallel.

Figure 8 shows the result of running LSAT's timing analysis, i.e. a Gantt chart of the four machine cycles in Figure 7. The Gantt chart shows the actions performed by the system's peripherals over time. In this Gantt chart, the low blocks represent resources being claimed by activities and the high blocks the actions and movements executed by the peripherals; arrows represent the sequence dependencies between the peripheral actions, the peripheral movements, the resource claims and the resource releases. The blocks' colors correspond to the involved activities. As the indexing activities (red) of the wafer table, transfer, product holder do not share resources, they run in parallel. The same holds for the pick-up (blue) and attach (green) activities. This corresponds to the pipeline behavior shown in Figure 2.

```
activities {
    IndexTransfer
    IndexWaferTable
    IndexProductHolder
    Pickup[1]
    Attach[2]

    IndexTransfer
    IndexWaferTable
    IndexProductHolder
    Pickup[2]
    Attach[1]

    IndexTransfer
    IndexWaferTable
    IndexProductHolder
    Pickup[1]
    Attach[2]

    IndexTransfer
    IndexWaferTable
    IndexProductHolder
    Pickup[2]
    Attach[1]
}
```

Figure 7: ADAT3-XF logistics model.

4.2 Model Validation

We based the modeling described in Section 4.1 on available documentation and knowledge of ITEC's domain experts. As it is difficult to obtain a complete system specification from these ambiguous and incomplete sources of information, it is essential to validate the model before using the model to predict the system's productivity. We validated the LSAT model by comparing the model's execution traces to the system's. We did this using TRACE4CPS[4] (Hendriks et al., 2017). This is a tool, which is used inside LSAT to visualize model execution traces, but which also allows analysis and comparison of execution traces as well as run-time verification.

Figure 9 shows a small part of the system's execution trace, after translating it to the TRACE4CPS

[4]https://www.eclipse.org/trace4cps/

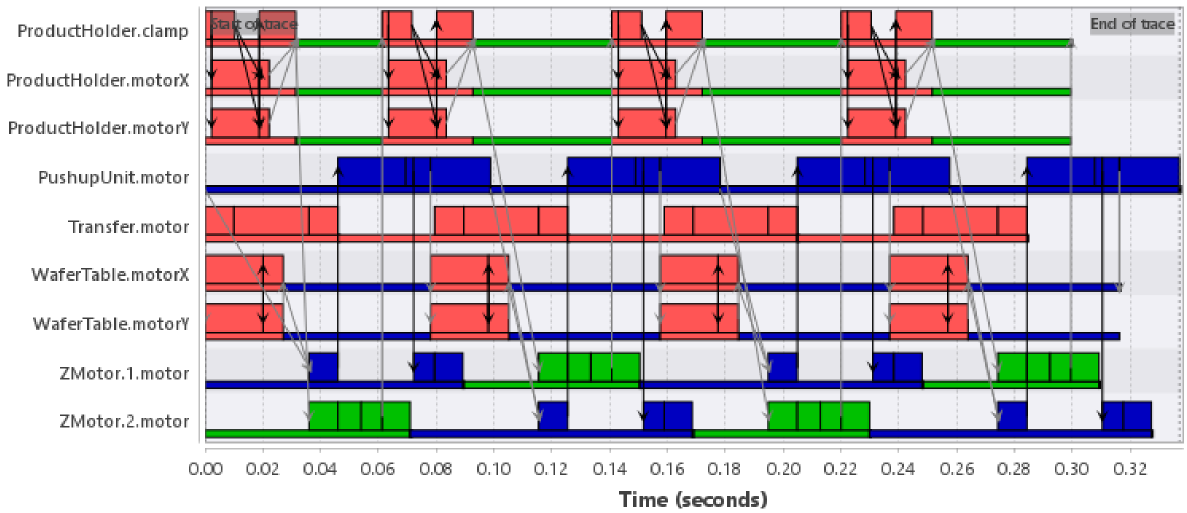

Figure 8: Gantt chart of four ADAT3-XF machine cycles.

format using an ITEC-proprietary tool. A swim lane shows either a continuous signal representing a motor position or discrete events. Figure 10 shows the corresponding part of the model's execution trace: multiple executions of the actions of the push-up unit when picking up a die (the blocks in the PushupUnit.motor swim lane in Figure 8).

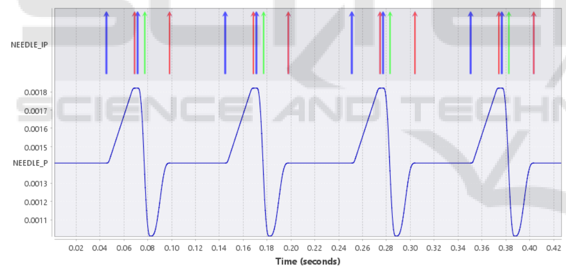

Figure 9: System execution trace (in TRACE4CPS).

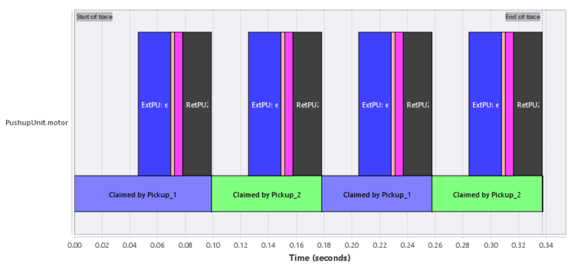

Figure 10: Model execution trace.

Clearly, one cannot directly compare the execution traces in Figures 9 and 10; the information in the system's execution trace is more detailed than that in the model's. To unify the abstraction level of the execution traces, we have created and applied a transformation language based on TRACE4CPS' run-time verification functionality. We use the system execution trace in Figure 9 as an example. This figure shows the events and signals corresponding to a needle pushing a die from the wafer onto a Z-motor's collet. The top swim lane shows the needle motor events, the bottom swim lane the position of the needle. Red and blue events correspond to starting and stopping the motor; green events are sent when a certain position is reached. The needle performs a sequence of four actions: (1) extension, (2) push up, (3) partial retraction, and (4) full retraction. Using the rules in Figure 11, we derive the start and end of these actions from the execution trace in Figure 9. The rules combine an event color and a needle position to determine the start and end of the model's needle actions. Figure 12 shows the resulting execution trace after applying the rules in Figure 11.

```
transformation needle_actions {
    interval: [-0.4, 0.0] s

    signal pos : {'signal'='NEEDLE_P'}

    def blue:  {'signal'='NEEDLE_IP', 'color'='half_blue', 'type'='spike'}
    def red:   {'signal'='NEEDLE_IP', 'color'='red', 'type'='spike'}
    def green: {'signal'='NEEDLE_IP', 'color'='green', 'type'='spike'}

    claim-spec extend:
        (blue and pos <= 0.0015), (red and pos >= 0.0015), {
        'resource'='PushupUnit', 'peripheral'='motor', 'action'='extend'}
    claim-spec pushup:
        (red and pos >= 0.0015), (blue and pos >= 0.0015),
        {'resource'='PushupUnit', 'peripheral'='motor', 'action'='pushup'}
    claim-spec retract1:
        (blue and pos >= 0.0015), (green and pos <= 0.0015),
        {'resource'='PushupUnit', 'peripheral'='motor', 'action'='retract1'}
    claim-spec retract2:
        (green and pos <= 0.0015), (red and pos <= 0.0015),
        {'resource'='PushupUnit', 'peripheral'='motor', 'action'='retract2'}

    shift-to: 0.0 s
    save-with-suffix: norm
}
```

Figure 11: Transformation specification.

The Gantt chart in Figure 12 strongly resembles the needle actions shown in Figure 10. When apply-

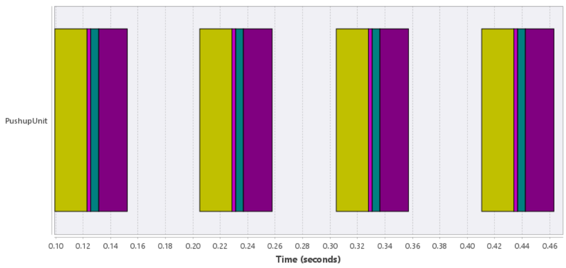

Figure 12: Transformation output.

ing another transformation on the Gantt chart in Figure 10, we obtain a Gantt chart with only the model's needle actions (the high blocks in Figure 10), leaving out the resource claims (the low blocks in Figure 10). Both transformed Gantt charts can be compared in TRACE4CPS by opening them simultaneously. This is shown in Figure 13. One can easily see that the order of the needle actions are identical in both Gantt charts. For more complex Gantt charts, this may be less obvious. Then, one can TRACE4CPS' trace comparison for this sequence check. In addition, TRACE4CPS' timing analysis was used to show that the durations of the actions was identical.

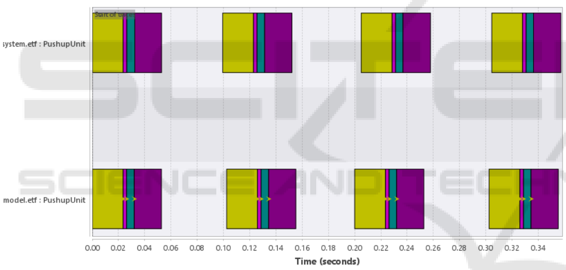

Figure 13: Gantt chart comparison.

Despite the matching action sequence and matching action durations, the total duration of the system's execution trace (top) is slightly longer than the model's (bottom). In other words, the model was valid on an individual action level, but not on an overall level. The difference of the total duration was caused by a delay in the system: the system's motion controller introduces a small delay between issuing a start signal and the actual start of the motor. Because we did not include this delay in the model, this caused a delay which slowly builds up over time, as one can see in Figure 13.

In subsequent models, we captured the delay between the start signal and the motor start in two manners: (1) for actions specified using motion profiles, a motion profile was introduced including a short start delay and (2) for actions without motion profiles, a motion controller delay action was introduced. With these changes, the system and model had both the same dependencies and the same durations. In other words, we obtained a valid model, i.e. a model whose behavior accurately matches the system's behavior. Such a model can be used as a starting point for exploring yet-to-be-developed variants.

5 PRODUCT FAMILY MODELING

There are many variants of ADAT3-XF die bonders. The systems in this family share resources; e.g. the die transfer, the Z-motors and the wafer table are used in most system variants. Similarly, activities may be shared by multiple variants; if two variants share the wafer table, they may also share the pick-up activities and if they share the same product holder, they may also share the attach activities. On the other hand, there are differences with respect to the number of dies in the system's pipeline, the quality inspections being performed, the dimensions of system components, and the speeds of the peripherals.

To facilitate the reuse and variability of the ADAT3-XF system family, we created a *master model* in LSAT and a corresponding *master transformation* in TRACE4CPS. The master model is a collection of LSAT models providing a library of LSAT model elements; the master transformation is a library of transformation rules that can be used to translate model and variant execution traces to a unified form.

The master model describes the resources and peripherals of all ADAT3-XF system configurations including the peripheral actions and these actions' settings. The master model also specifies all configurations' activities. Because of the desired reuse, the master model's activities became more fine-grained than those of the individual variants. For instance, the (simplified) variant's pick-up activity shown in Figure 6 was decomposed into separate activities of the Z-motors and the push-up unit with synchronization resources to keep the desired logistic process. One of these activities is shown in Figure 14. It describes how the extension of a Z-motor to the wafer table. Beside the Z-motor resource, it involves two synchronization resources to avoid other activities from starting too early.

To facilitate reuse of model elements, the master model was made in a modular fashion following the component structure of the system variants. Figure 15 visualizes the master model structure in relation to the product family structure. Each system variant has a dedicated logistics model with the appropriate activity dispatching sequence. A logistics model of an ADAT3-XF system variant uses activities from the activity models describing the activities of the variant's

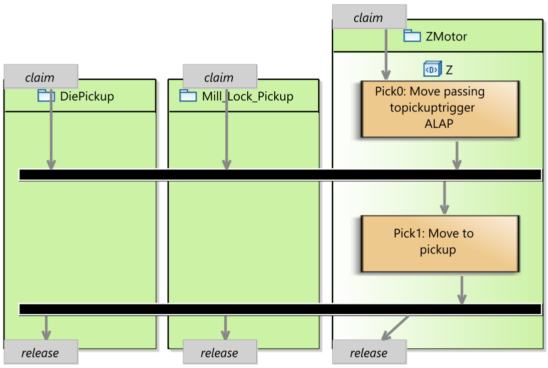

Figure 14: Reusable activity element.

components. These activity model uses the machine models of the resources used in their activities. Machine models do not use other types of models.

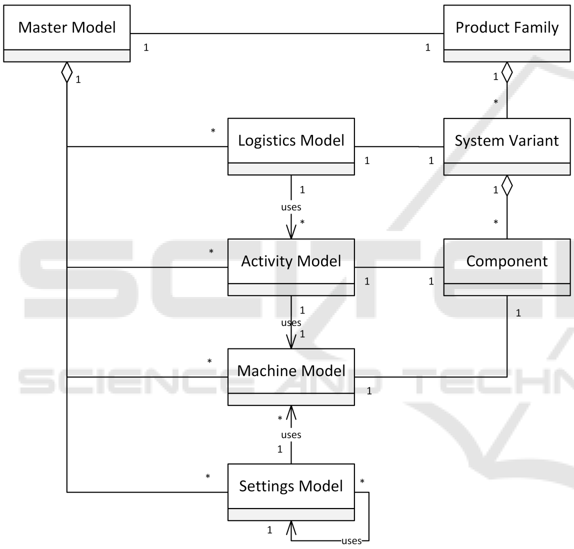

Figure 15: Master model structure.

Per system component, the master model contains a single machine model and a single activity model. The former describes the component in terms of resources and peripherals; the latter describes the activities with this component as the main resource. For instance, the machine model of the push-up unit describes the motors of the push-up unit and the corresponding activity file describes the needle's extension, push-up and retraction activities.

Because of the diversity of settings, modularity and reuse of settings models proved more complex than for machine and activity models. Settings models are used to determine the duration of peripheral actions. Durations are either specified as fixed values or they are derived from a description of a movement. For movements, settings models contain information about distance, velocity, acceleration, and jerk. Both types of settings can be specified as (closed) expressions containing settings in other settings models.

The usage relation of the master model's settings models is shown in Figure 16. This is a layered structure with settings models for machine, motor, and physical constants forming the lowest layer. Machine constants settings models describe the physical characteristics of available resources, motor constants settings models include the parameters of the peripherals' motion profiles, and physical constants settings models include the variables that are not coupled to a resource or peripheral. The second layer contains the run configuration settings models that are specified in terms of the lowest layer's constants. The run configuration settings are used in the settings models of the peripheral actions, which are combined in a single root settings model of a system variant.

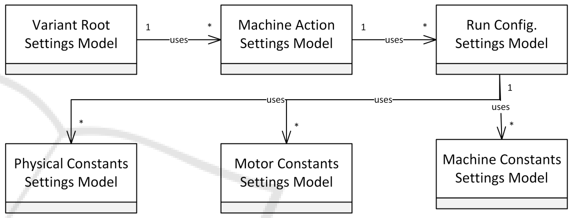

Figure 16: Settings models structure.

6 LESSONS LEARNED

In Sections 4 and 5, we have outlined our work of introducing productivity modeling in brown-field product family development. This section describes the lessons that we have learned while developing productivity models of a family of die bonder systems. We distinguish modeling lessons (see Section 6.1) and organizational lessons (see Section 6.2).

6.1 Modeling Lessons

Parameterization. The ADAT3-XF systems are high-productivity machines that handle tens of thousands of products per hour. To obtain such high productivity, the systems support a pipelined execution (see Figure 2). To allow parallel handling of the products in the pipeline, the system contains multiple resources of the same type. For instance, multiple Z-motors are used to pick up one die and, in parallel, attach another. To keep the model of a system variant concise, we learned that parameterization of resources and activities is essential. Parameterized activities use parameterized resources and are used multiple times in a dispatching sequence (see Figure 7). Without parameterization, a system variant model would be much longer and its maintenance would involve a large effort as one needs to keep all instances consistent. In the context of a family of system variants, parameterization becomes even more important as it involves even more reuse.

Modularity vs. Understandability. To maximize reuse in product family modeling (see Section 5), we noted that the model elements became smaller. This especially holds for the activities describing the system behavior. By dividing one activity into multiple smaller activities, one has to take care of the dependencies that were captured by the original activity. In our use case, this meant the introduction of additional synchronization resources in the decomposed activities that replace the cut precedence constraints in the original activity. The smaller activities increase model element reusability, but we observed that the introduced synchronization resources reduce model understandability and maintainability. To improve the situation, we advice to use another synchronization concept of LSAT: activities can synchronize either via shared resources or via events. The latter can be used to signal that e.g. actions have completed. These events are more natural to system developers than the synchronization resources.

Variation Modeling. By making the step from individual systems to a family of systems, we learned that tools intended for individual systems cannot just be used for families of systems.

In our use case, we used LSAT to create a master model consisting of reusable model elements. LSAT's support for modularity proved adequate for its machine and activity models. However, we experienced that this is not true for its settings models. To allow reuse of LSAT's settings models, we needed a complex hierarchy of settings models (see Figure 16). In this hierarchy, elements influencing the whole system are on the lowest level. In order words, the hierarchy of settings models does not follow a normal hierarchy pattern, in which the lowest levels have the smallest influence.

Our use case showed the difficulties that one faces when introducing a model-based way of working in a brown-field development situation. We learned that we needed to address many challenges to come to a working solution. We could have overcome some of the challenges by handling system variability differently: e.g. we could use variability modeling tools like pure::variants[5] or FeatureIDE[6] to describe all possible variability and to generate variant productivity models from.

[5]https://www.pure-systems.com/

[6]https://www.featureide.de/

Model Validation by Abstraction. During model validation, we faced the challenge that the systems' execution logs are more detailed than the models'. The systems' execution logs contain all motor start and end signals, but they do not show which action starts or ends. To overcome this difference, we raised the abstraction level of the systems' execution log using a set of transformation rules similar to those in Figure 11. After applying these rules to a variant's execution trace and the corresponding model execution trace, we could use TRACE4CPS' comparison functionality to validate model correctness.

Reuse of Validation Rules. We learned that reuse of these transformation rules is not straightforward for different instances of the same system. The distances traveled by the motors of a system differ per system, even if they have the same configuration. As the motor positions are used to do model validation, we observed that one needs to carefully consider the *magic numbers* in the transformation rules. These should match the characteristics of the system used for model validation. This is even more challenging for a family of system variants, as their magic numbers may also vary across different variants.

Incremental Model Validation. Before we could use the models to predict the productivity of system variants, they needed to be validated. We learned that an incremental model validation approach was very effective. The transformation rules used in the master transformation guaranteed the correctness of (the timing of) the model elements. Having the model elements validated allowed us to focus on the validation combinations of model elements. As the second step, we validated the timing and order of the actions of each peripheral using TRACE4CPS' trace comparison functionality. Finally, we used the same functionality to validate the timing and order of the actions of the entire model.

We noted that over time the differences between the system and the model became more and more subtle, requiring more and more domain knowledge. A clear example is the delay between start signal and motor start discussed in Section 4.2. Although this is visible for individual peripherals (see Figure 13, we only identified the cause after observing the timing difference throughout the entire execution trace.

6.2 Organizational Lessons

System and Model Co-Evolution. In this paper, we described how we introduced a master model and a corresponding master transformation for the family of ITEC's ADAT3-XF die bonders. This required a significant effort, but we also realized that this effort does not stop here. The family of die bonders will change over time: faster variants of existing configurations will be introduced as well as variants applying new technology. To avoid that the efforts made to create the master model and model transformation are wasted, the master model and model transformation should co-evolve with the product family. This requires embedding the master model and model transformation in the way of working of the organization. Ideas to achieve this include (1) putting the master model and master transformation under version control just like the source code, (2) making the model and transformation part of the system documentation, and (3) annotating the system's source code with knowledge about the master model and master transformation. The latter would also greatly simplify the specification of transformation rules.

System Changes for Modeling. When specifying the rules to validate our models, we observed that some rules were quite complex and hence hard to reuse. Instead of specifying complex model validation rules, we think it is more convenient to adapt the system for the purpose of predictive modeling. E.g. the events in a system execution log could be extended with information about the corresponding model action. By extending the system's execution logs with knowledge of the corresponding model, (1) there is an explicit mapping between the system and the corresponding variant model, (2) transformation rules become simpler and the master transformation more maintainable, and (3) model validation becomes (more) straightforward.

Democratization of Knowledge. In Section 4, we explained how we modeled individual system variants. To create valid LSAT models for ADAT3-XF systems, we started modeling an existing system variant. We created the variant model by capturing expert knowledge and comparing model and system execution traces. The model creation and validation explained in Section 4 showed that our approach was an iterative one: the differences identified by comparing Gantt charts revealed information that was not yet captured by the model. It was challenging to capture all dependencies, especially those between actions of peripherals of different resources. By eventually obtaining a validated model, the model and the corresponding transformation rules became a non-ambiguous source of domain knowledge. ITEC engineers saw this as a valuable asset to transfer (implicit) domain knowledge in experts' heads to new employees.

Tool Usability. In our use case, we have used LSAT and TRACE4CPS to introduce productivity modeling for a family of die bonder systems. Both LSAT and TRACE4CPS use textual domain-specific languages (DSLs) to specify model elements. The intended users of of LSAT and TRACE4CPS within ITEC have a background in mechanical engineering. We found out that they are not familiar with textual interfaces. Creating instances of textual DSLs was said to feel like software development. A way to overcome this usability issue is to develop (possibly graphical) company-specific tooling that hides the perceived complexity of the existing languages.

Model-Based System Validation. In this paper, we have used transformation and comparison of execution traces to validate our productivity models. A validated model can be used to explore the behavior of yet-to-be-developed system variants (Parappurath et al., 2013). At the end of this exploration, the model provides a (partial) specification for the system. In other words, the role of model changes from descriptive to prescriptive. We realized that the execution trace comparison techniques can also be used to validate system behavior: by comparing unified system and model execution traces, one can validate whether the system's behavior is as the model predicts/prescribes.

7 CONCLUSION

In this paper, we have introduced a model-based way of working in brown-field product family development. The paper presents two results: (1) we developed (a) a master model, i.e. a library of model elements matching the elements of the product family and (b) a master transformation, i.e. a library of transformation rules to transform system and model execution traces to a unified form, and (2) we reflect on the process of introducing a model-based way of working by presenting experiences and lessons learned.

The use case shows that it is feasible to introduce a model-based way of working in brown-field product family development. It also shows that model-based development of a product family is significantly more complex than for an individual system. Many challenges need to be addressed: for a successful introduction of a model-based way of working, changes are required both on the system development side and on the modeling side. Some of these challenges are purely technical, but most (also) have a significant organizational aspect: to maintain a model-based way of working on the long run, changes are required in the system and in the organization.

The use case's master model and master transformation introduce the benefit of being able to predict the productivity of yet-to-be-development system variants. The use case shows that the benefits of model-based way of working are not limited to this. Beside the prediction benefit, our use case reveals two additional benefits of the master model and master transformation: (1) they capture knowledge in experts' heads and (2) they provide an additional means to validate systems.

ACKNOWLEDGEMENTS

The research is carried out as part of the Bright program under the responsibility of TNO-ESI with ITEC as the carrying industrial partner. The Bright research is supported by the Netherlands Organisation for Applied Scientific Research TNO.

REFERENCES

Czarnecki, K. and Eisenecker, U. W. (2000). *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, Reading, MA, USA.

Feng, Y., Dreef, K., Jones, J. A., and van Deursen, A. (2018). Hierarchical abstraction of execution traces for program comprehension. In *26th Conference on Program Comprehension (ICPC '18)*, pages 86–96, Gothenburg, Sweden.

Gad, R. (2017). Improving packet capture trace import in Trace Compass with a data transformation DSL. In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC 2017)*, volume 2, pages 7–12, Turin, Italy.

Grönniger, H., Krahn, H., Pinkernell, C., and Rumpe, B. (2008). Modeling variants of automotive systems using views. In Klein, T. and Rumpe, B., editors, *Modellbasierte Entwicklung von eingebetteten Fahrzeugfunktionen*, number 2008-01 in Informatik-Berichte, pages 76–89. Technische Universität Braunschweig, Braunschweig, Germany.

Hendriks, M., Verriet, J., Basten, T., Theelen, B., Brassé, M., and Somers, L. (2017). Analyzing execution traces: Critical-path analysis and distance analysis. *International Journal on Software Tools for Technology Transfer*, 19(4):487–510.

Hoffman, C. M. and Joan-Arinyo, R. (1998). CAD and the product master model. *Computer-Aided Design*, 30(11):905–918.

Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., and Peterson, A. S. (1990). Feature-oriented domain analysis (FODA) feasibility study. Technical report, Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, PA, USA.

Meixner, K., Rabiser, R., and Biffl, S. (2019). Towards modeling variability of products, processes and resources in cyber-physical production systems engineering. In *23rd International Systems and Software Product Line Conference (SPLC '19)*, volume B, pages 49–56, Paris, France.

Parappurath, V. V., Voeten, J. P. M., and Kotterink, K. C. (2013). Calibration error bound estimation in performance modeling. In *2013 Euromicro Conference on Digital System Design*, pages 97–102, Los Alamitos, CA, USA.

Reiss, S. P. (2005). Dynamic detection and visualization of software phases. In *Third International Workshop on Dynamic Analysis (WODA '05)*, pages 1–6, St. Louis, MI, USA.

Sandberg, M., Tyapin, I., Kokkolaras, M., Isakasson, O., Aidanpää, J.-O., and Larsson, T. (2011). A knowledge-based master-model approach with application to rotating machinery design. *Concurrent Engineering*, 19(4):295–305.

Tawhid, R. and Petriu, D. (2008). Integrating performance analysis in the model driven development of software product lines. In Czarnecki, K., Ober, I., Bruel, J.-M., Uhl, A., and Völter, M., editors, *MODELS 2008: Model Driven Engineering Languages and Systems*, pages 490–504. Springer, Berlin, Heidelberg.

van der Linden, F. J., Schmid, K., and Rommes, E. (2007). *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Springer-Verlag, Berlin, Heidelberg.

van der Sanden, B., Blankenstein, Y., Schiffelers, R., and Voeten, J. (2021). LSAT: Specification and analysis of product logistics in flexible manufacturing systems. In *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, pages 1–8, Lyon, France.

Verriet, J., Dankers, R., and Somers, L. (2018). Performance prediction for families of data-intensive software applications. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering (ICPE '18)*, pages 189–194, Berlin, Germany.