MODEL BASED CONTROL SOFTWARE SYNTHESIS FOR PAPER HANDLING IN PRINTERS

Chitralekha Pillai
Embedded Systems Institute
P.O. Box 513, 5600 MB Eindhoven
The Netherlands
chitralekha.pillai@esi.nl

Ronald Fabel
Océ-Technologies B.V
P.O. Box 101, 5900 MA Venlo
The Netherlands
ronald.fabel@oce.com

Lou Somers
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven
The Netherlands
l.j.a.m.somers@tue.nl

KEYWORDS
Model based engineering, software synthesis, multidisciplinary design, domain specific languages, behavior specification.

ABSTRACT

Control software is an integral part of new complex electromechanical systems, such as professional high speed printers. The development of these multidisciplinary products involves a number of iterative and incremental cycles of prototype creation. Automatically generating control software is a big leap in rapid prototyping of these products. A model based approach is an effective way to manage the complexity as well as to enable new insights in terms of innovations and technology risks. Models are widely used for understanding and designing the system, but not yet for control software generation. The main challenge in generating complete and executable control software is the difficulty in specifying the behavior at the abstraction level of a generic modeling language. By specifying the static and behavior design information in domain specific models, the level of abstraction is raised to using the concepts and rules of a specific problem domain. Focusing to a narrow domain enables automation to the level of full code generation. This industrial case explains how a domain specific modeling approach was applied to paper handling in printers to generate the control software. The promising result is a motivation to develop this approach further for other printer modules.

INTRODUCTION

Productive printing systems have evolved into innovative and complex multi-disciplinary products. The amount of software control in printers is increasing because of the extensive functionality they offer. As a result, a lot of development effort is shifted towards development of control software. Models provide an effective way to develop large, reliable, real-time software systems because they help engineers to manage complexity (Heemels et al. 2006). A model-based synthesis of control software is generating the control software directly from the models. The generated code can be linked with the underlying platform code and compiled to a finished executable without additional manual effort. The key to this level of code generation is to specify the multidisciplinary design information at the right abstraction level. A domain specific approach is ideal because it fits to the domain needs and relates to the required behavior closely. A domain specific code generator can then translate the high level specification to code.

Automating the control software development opens up opportunities for quicker feedback on design decisions. It works well when software and hardware requirements and specifications are changing during development of a product. The ability to synthesize software from models helps to ensure the consistency of design information throughout the development lifecycle. This ensures that the design specification matches the implementation and thus tremendously decreases the development effort and time as well as improves the quality of the products. The role of embedded software developer now changes to that of a domain expert who can focus on multidisciplinary aspects of the design. A common overview of the essential multidisciplinary design information represented in a domain model serves as an excellent and necessary means to manage design information across different activities and different disciplines during the development cycle.

In Figure 1 we visualize how control software synthesis facilitates the development of complex systems like printers using a model-based approach. The potential benefits of automation in the development process can be observed.

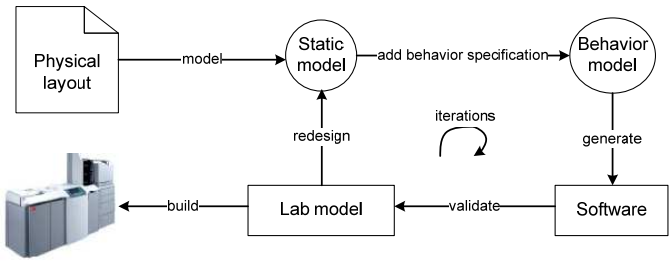

Figure 1: Model Based Control Software Synthesis: Workflow

The development starts with envisioning the printer and coming up with a model of the physical layout. The resulting multidisciplinary design information model represents the machine description at an abstract level which can be communicated across disciplines. If this is a model understood by all disciplines, the consistency of the design information can be maintained throughout the development (Schindler 2008). Based on this static model and a behavior specification language, the functional requirements can be expressed as a behavior model. This behavior model along

© EUROSIS-ETI

with the underlying multidisciplinary design information model can be used to automatically generate the control software. This generated software has I/O control interface calls to control the electromechanical parts of the printer. The generated software can be validated in the lab model. For a redesign, another iteration is started. When the lab model evolves to a sufficient quality, it is used to build the actual machine.

It can be observed that automatic generation enables consistency of the model and the generated software. The development time can be shortened by quick design decision verification and reduced development effort by automating repetitive tasks.

Being free from the manual creation and maintenance of source code significantly improves developer productivity. The reliability of automatic generation compared to manual coding will also reduce the number of defects in the resulting programs and thus improves quality. This automated transformation process is often referred to as "correct-by-construction," as opposed to conventional handcrafted "construct-by-correction" software development processes that are tedious and error prone (Schmidt 2006).

The remainder of the paper explains the details of the case study. First, the domain overview is explained, which introduces the paper path in a printer. This helps in understanding the essential multidisciplinary design information of a paper path. These concepts are mapped to a static design information model of the paper path. Then the domain concepts used to define the sheet behavior specification language and the resulting specification are described. Finally, the software synthesis setup is explained.

PAPER HANDLING IN PRINTERS

A paper handling module consists of both hardware and software responsible for sheet movement inside the printer, from a Paper Input Module to a Finisher. These different elements in the printer (such as mechanical parts, electrical connections and software) work together to move the paper.

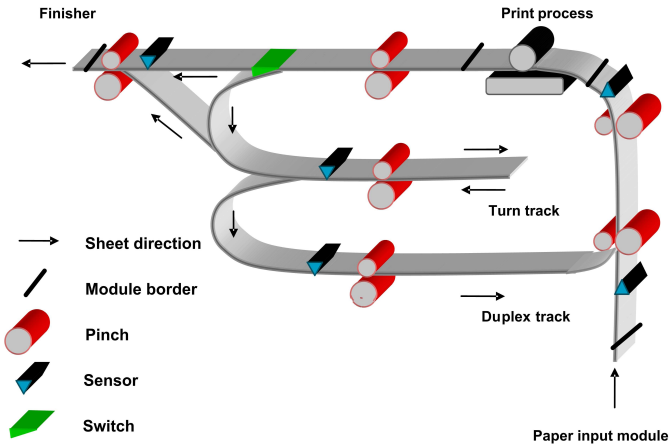

Figure 2: Conceptual Overview of a Paper Path

The track is in the form of iron plates that lead the sheet through a predefined shape of movement. The transportation of sheets on the track is realized by pinches. Each pinch consists of two pinch rollers that hold and move the sheet by means of friction. Pinches are directly or indirectly (through clutches and/or pulleys in between) driven by motors. Note that one motor can drive more pinches simultaneously. The distance between pinches should not be larger than the length of the smallest sheet which the printer is specified to handle, otherwise the paper sheet loses its speed. A sheet may, however, be transported by multiple pinches simultaneously. Paper path sensors detect the sheet. They can be used to track the real position of sheets in the paper path by sheet edge detection. The switch is intended for changing the direction of sheets towards the finisher or the turn track. As it can be seen in the layout example in Figure 2, there may be a turn track for turning the sheet (in case of duplex printing) and multiple tracks to the Finisher, an upper track and a lower track.

The control software is responsible for controlling these various mechanical and electrical parts and delivering sheets of the right format (wrong formats result in an error), with a specified alignment, position, and skewness, with a required speed and temperature on a scheduled time to the Print Process (where the actual printing takes place), and transportation of sheets from the Print Process to the Finisher.

MULTIDISCIPLINARY DESIGN ASPECTS

Though on first sight a paper path seems simple, the realization of sheet transportation involves many subtle trade-offs such as cost, speed, productivity and space. The following (strongly interconnected) aspects play a role in paper handling module design:

Spatial layout: the shape of the paper path. The lengths of the tracks in the path and positions of mechanical parts such as pinches and sensors become the characterizing parameters.

Timing: sheet position in the paper path at a given time. This has to be synchronized with other related functions in the printer, like the printing process.

Drive map: connection of pinches to motors, either directly or by means of clutches and pulleys. Such a connection configuration dictates the constraints on using pinches to implement timing design. For example: if two pinches are connected to the same motor, one cannot turn in another direction than the other.

Corrections: compensation for anticipated issues. Nominal (planned) corrections are part of the alignment process. Corrections compensate for small tolerances in sheet movement caused by different paper types, slipping of pinches, tolerances in mechanical production, or simple wear and tear of mechanical parts.

Exception Handling: takes care of errors that can occur during sheet movement. For example: a sheet may have a crease at the side, which causes it to get stuck in a paper track. Such an error must be handled accordingly (e.g. by moving the sheet forcefully to a certain location where an operator can reach it and remove it).

© EUROSIS-ETI

STATIC DESIGN INFORMATION MODEL

The highest level description element in the model is chosen to be the "Machine", which consists of several aspects. The multidisciplinary design aspects explained above are mapped to these aspects. In this case study, the static design information model consists of only relevant aspects related to the paper path:

Parts: physical things in the machine. Parts describe properties of machine components such as pinches, sensors, motors, that are essential as static design information.

Topology: describes the essence of the spatial layout, which serves as a spatial description including the relevant parts identified as point of interests (POI) in the layout. The descriptive elements of the topology are segments. A sequence of segments specifies a route. Points of interest (POI) can be any points which are used for specifying any important issue onto the topology.

Appearance: describes the position of all interesting physical elements in the machine. It is expressed as co-ordinates in the machine layout.

Drive map: contains the connection type and gear from motor control interface to parts.

I/O aspects: contains input and output lines of parts, required for I/O layer calls.

SHEET BEHAVIOR SPECIFICATION LANGUAGE

The sheet movement in the paper path is termed as "sheet behavior". The control software in the paper handling module is responsible for controlling the various mechatronic devices to achieve the correct sheet behavior. This is crucial since the paper has to be strictly timed with respect to other functionalities of the printer, without any collisions. The timing designer is responsible for describing the timing requirements for different kinds of print jobs to realize the corresponding sheet behavior on the machine. Each sheet undergoes velocity changes according to the timing requirements. Often these velocity changes are derived and calculated from the spatial layout. The timing designer can do simple calculations in an excel sheet to come up with references to when and where the sheet velocity has to be changed. Hence, the sheet behavior can be represented in terms of sheet velocity (Beckers et al. 2007). The corresponding control software is a sequence of control actions for the mechatronic devices to achieve this sheet velocity profile.

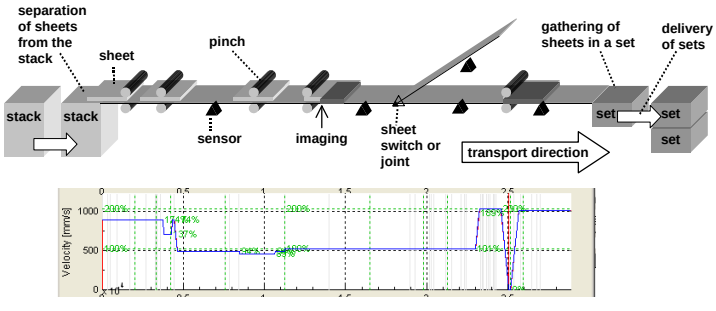

Figure 3: A Typical Sheet Velocity Profile in a Paper Path

The domain specific language consists of a set of directives which helps to specify the sheet behavior in the paper path. These directives were derived by classifying the control to four situations as explained below.

The arrival of a sheet

The information regarding when to expect a sheet is available from the print job definition. This is the starting directive of sheet behavior specification:

SheetArrival (arrivalPoint, arrivalTime, speed)

Here, *ArrivalPoint* indicates the reference position associated with the synchronization time specified by the driving software. This is included as POI in topology. *ArrivalTime* is the synchronization time to expect the arrival of a sheet at the reference position from the paper stack. As mentioned above, this is obtained from the print job. *Speed* is the speed (in mm/s) at which the sheet reaches the reference position.

Sheet velocity changes

The timing designer wants to specify the position of a sheet at any point in time. This is done by calculating the required sheet velocity profile. The drive map of a pinch contains information about the maximum allowed acceleration for that pinch given the mechatronic design. As a future work, the behavior specification can be extended to include a constraint check to make sure the required acceleration is within the maximum acceleration given in drive map.

SheetVelocityChange (Reference time/position, Offset time/distance, speed, acceleration)

Reference Time/Position is the reference based on which the sheetvelocitychange point is based. *Offset Time/Position* is the positive offset to add to the reference to get the exact point. *Speed* is the desired speed value at that point. *Acceleration* is the required acceleration for the speed change given the mechatronic design.

Sensor controls

A sensor will give the software momentarily information of a position/time pair of the leading or trailing edge of a sheet. The absolute position of the sensor is found in topology, the absolute time is available in the software after the sensor senses the sheet. The behavior specification can introduce a reference variable for this captured time which can be used in other directives to specify timing of actions.

DetectSheet (sensor, detectionwindow, time reference)

Sensor indicates the sensor which has to detect the sheet. *Detectionwindow* is the window in which the sensor expects the sheet. If the sheet is not detected within that window, an error should be flagged. *Time reference* indicates the time captured during this action. This can be used as a time reference for usage in other directives.

Sometimes a timing designer wants to specify a speed change point based on the sheet position in the paper path. In order to be able to execute a directive which is based on a sheet position, it is needed to "know" the sheet position. If

© EUROSIS-ETI

the sheet position is not made available by the driving software, any position reference which is not on a sensor position must be calculated by using the time and the velocity profile. In such a case, the calculation can already be done in an excel sheet and the directive can use a time reference. So a position reference can only be used if the sheet position is available. This is done by a position capture at a sensor. The position capture sensors can capture the position of sheet in terms of motor steps. The directive to do so also introduces the position reference identifier that refers to the specific position following service in the driving software.

TrackSheetPosition (sensor, pinch, position reference)

Sensor indicates the sensor point from which the position tracking starts. *Pinch* is the pinch (and thus motor) to be used for this purpose. *Position reference* is the identifier to be used in further directives as the position reference.
Note that this directive is not purely a sheet behavior directive. It enables the timing designer to specify a position reference according to his requirements, which can be used in other directives.

Controlling pinches and switches when there is no paper

When the sheet is approaching a pinch, the pinch has to be prepared to receive the sheet so that there is a smooth takeover of sheet. If the receiving pinch is not running at sheet velocity, the sheet movement is not smooth. We need to control the pinch/motor in this situation.

PinchPrepare (Pinch, Reference time/position, Offset time/distance, speed, acceleration)

Pinch denotes which pinch has to be prepared to receive the sheet. *Reference Time/Position* is the reference based on which the pinchprepare point is based. *Offset Time/Position* is the offset to add to the reference to get the exact pinchprepare point. *Speed* is the desired speed of the pinch. Acceleration is the maximum allowed acceleration for that pinch given the mechatronic design.

The directives explained above can be used to specify the sheet behavior in a modular manner. The sheet behavior specification contains a sequence of these directives with the appropriate parameters. There are references to the multidisciplinary static design information model. Each directive corresponds to a number of control actions to be performed at specific moments. Hence, the generated control software can be visualized as a state machine, with control actions being executed when corresponding time or position guards become true.

SOFTWARE SYNTHESIS

The aim of model based control software synthesis is generating complete finished code for what has been modeled. The software synthesis approach applied in this case is not based on formal methods (Bertens et al. 2008, Markovski et al. 2010), but on domain specific concepts. Every domain contains its own specific concepts and correctness constraints. By working at this abstraction level, the behavior of a system can be precisely specified, which is difficult in terms of a formal method or a generic modeling language (Fowler and Scott 1999). This level of specification is crucial for full code generation. The rules of the domain can be included as constraints, ideally making it impossible to specify illegal or unwanted design models (Kolovos et al. 2006). This results in an expressive yet bounded model within the domain that can be used for code generation (Luoma et al. 2004).

The software synthesis system consists of domain specific models, a domain specific code generator and a domain framework (Kelly and Tolvanen 2008). The generator extracts the information from the models and transforms it into code. This translation is done based on the rules of the domain and how the generated output should look like. The domain framework provides the interface between the generated code and the underlying platform. The generated code is linked with the framework and compiled to a finished executable without any additional manual effort. The generated code is thus simply an intermediate by-product on the way to the finished product, like object files in C compilation.

Let us see how this approach was applied in the printer case. In the printer embedded software architecture, there is a layer called controller which is responsible for mapping a print job to page commands. Each sheet gets a new page command. The page command contains information such as simplex/duplex printing, paper size and when to expect the arrival of the sheet. The sheet behavior for each sheet is based on this page command and the specification in the multidisciplinary design information model.

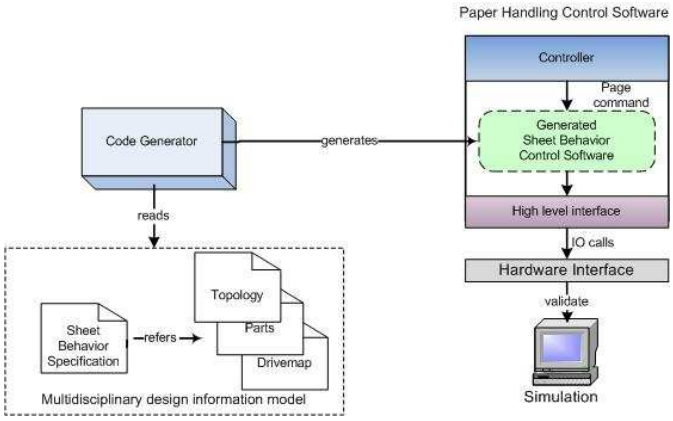

Figure 4: A Simplifed Representation of the Model Based Control Software Synthesis Setup

The code generator reads the multidisciplinary design information model which consists of both static and sheet behavior specification. Each directive in the sheet behavior specification means certain desired control behavior. The translation of a directive by the code generator involves determining which mechanical parts are responsible for the desired result and setting control commands to those parts at specific instants. The details of the mechanical parts are available in the static design information model. The specific instances can be deduced from the arguments in the directive.

© EUROSIS-ETI

For example, consider the directive

SheetVelocityChange(TurnPi,10,890,20000).

This means that the sheet velocity has to be changed to 890mm/s at an offset distance of 10mm from pinch with name TurnPi. Now the code generator has the information about which motor is driving the TurnPi. Based on the paper size, the code generator can also find out which previous pinches are still influencing the current sheet. The control commands have to be transmitted to all concerned motors for the desired speed change.

Thus the output of the code generator is a state machine with a sequence of control commands which gets executed when the specified moment has reached. In order to integrate this state machine to the underlying platform, a high level interface layer has been defined. This layer takes care of abstracting the hardware I/O calls to generated code level. Another advantage of this abstraction is the reuse of the code generator even if the underlying hardware changes.

There are no manual changes to be performed in the generated software. It is linked with the controller software layer and compiled to a finished executable which can be tested on a simulation environment before the actual machine is ready.

Evaluation / results

During this case study, the evaluation was carried out in a simulation environment (Software-In-the-Loop, SIL) of a real printer development project. The multidisciplinary design information model consisted of the static design information of the printer (layout, drive map, etc.) and the sheet behavior specification for a "simplex" print job. The specification started with a SheetArrival, followed by several SheetVelocityChanges and DetectSensor directives. The code generator now translated the specification to a state machine which comprised of all the necessary control commands for seven motors and four sensors which are involved in the simplex behavior. There were 26 state transitions automatically generated to achieve this behavior.

The generated executable code was tested in the SIL environment, which delivers a log showing the timing behavior of each sheet of paper. This log was verified with a timing model to see if the intended paper timing behavior matches with that of the generated code. The results showed that the timing deviation was less than 1ms. The velocity profile from the timing model matched with that from the generated code. The timing was also cross checked with the existing handcrafted code. The results proved the feasibility of full code generation from models, which matches with the performance of a handcrafted implementation that took a lot of time to build.

CONCLUSION

Automation of control software development has been proved feasible in the case of the paper handling function in printers. The concept was demonstrated by synthesizing control software for simplex sheet behavior in a real printer development project.

This research is a stepping stone to an improvement opportunity in the conventional design and development of control software. The multidisciplinary design information model guards consistency and evolution of the system design, and facilitates communication across different disciplines. In model based software synthesis, the focus shifts from solution domain to problem domain. Embedded software developers who have been working at a low level of abstraction, writing code, are motivated to work at a higher level of abstraction and develop the models in the problem domain from which code, tests, documentation, and so on, are generated automatically. Their role will change from that of crafting software and tediously repeating work for every iteration to a domain expert who actually adds more detail to a design, leaving the laborious activity of converting design decisions to working software to tooling, wherever this is sensible and saves effort and time. This leads to improved productivity, better product quality, hiding complexity, and leveraging expertise.

REFERENCES

Beckers, JMJ, W.P.M.H. Heemels, B.H.M. Bukkems, and G.J. Muller. 2007. "Effective industrial modeling for high-tech systems: The example of happyflow". In *Proceedings of the 17th Annual International Symposium of the International Council On Systems Engineering, INCOSE* (San Diego, USA, June 24-28).

Bertens, E., R. Fabel, M. Petreczky, D.A. van.Beek, and J.E. Rooda. 2009. "Supervisory control synthesis for exception handling in printers". In *Proceedings of the 10th Philips Conference on Applications of Control Technology* (Hilvarenbeek, Netherlands, Feb 3-4).

Fowler, M. and K. Scott. 1999. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley.

Heemels, W.P.M.H., E. van de Waal E., and G.J. Muller. 2006. "A multi-disciplinary and model-based design methodology for high-tech systems" In *Proceedings Conference on System Engineering Research, CSER* (Los Angeles, USA, April).

Kelly, S. and J.-P. Tolvanen. 2008. *Domain-Specific Modeling: Enabling full code generation.* John Wiley and Sons, New York.

Kolovos, D.S., R.F. Paige, T. Kelly, F.A.C. Polack. 2006. "Requirements for Domain-Specific Languages". In *Proceedings of the 1st ECOOP Workshop on Domain-Specific Program Development, DSPD 2006* (Nantes, France, July).

Luoma, J., S. Kelly, J.-P. Tolvanen. 2004. "Defining Domain-Specific Modeling Languages: Collected Experiences". In *Proceedings of the 4th OOPSLA Workshop on Domain-Specific Modeling, DSM'04* (Vancouver, Canada, Oct).

Markovski, J., K.G.M. Jacobs, D.A. van Beek, L.J.A.M. Somers, and J.E. Rooda. 2010. "Coordination of Resources using Generalized State-Based Requirements". To appear in *Proceedings of the 10th International Workshop on Discrete Event Systems.* (Berlin, Aug 30-Sep 1).

Schindler, E. 2008. "MoBasE: A Multidisciplinary Model Based Engineering Framework for Development of Production Printers". Technical Report, Stan Ackermans Institute, Eindhoven University of Technology.

Schmidt, D.C. 2006. "Model-Driven Engineering". *IEEE Computer* 39, No. 2 (Feb), 25-31.

© EUROSIS-ETI