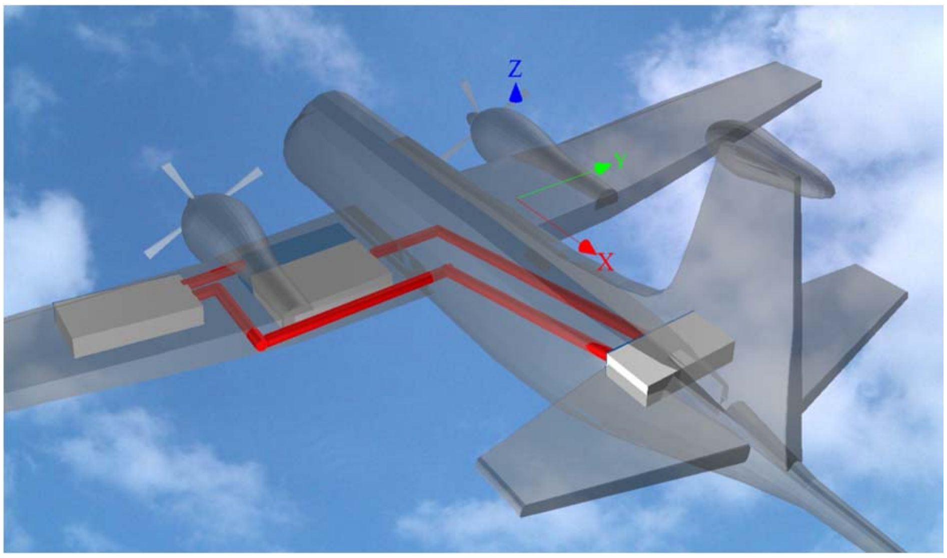
MULTI-DOMAIN MODELLING IN DESTECS AND PTOLEMY – A TOOL COMPARISON

Electrical and Computer Engineering
Technical Report ECE-TR-15



AARHUS
UNIVERSITY
DEPARTMENT OF ENGINEERING

DATA SHEET

Title: Multi-domain Modelling in DESTECS and Ptolemy
- a Tool Comparison

Subtitle: Electrical and Computer Engineering
Series title and no.: Technical report ECE-TR-15

Authors:
Sune Wolff, Department of Engineering – Electrical and Computer Engineering, Aarhus University

Ken Pierce, School of Computing Science, Newcastle University,

Patricia Derler, EECS Department, UC Berkeley

Internet version: The report is available in electronic format (pdf) at the Department of Engineering website http://www.eng.au.dk.

Publisher: Aarhus University©
URL: http://www.eng.au.dk

Year of publication: 2013 Pages: 19
Editing completed: October 2013

Abstract: Developing embedded systems with high performance and safety requirements is notoriously hard. It is not enough to have a thorough understanding of the control algorithms used, but a deep understanding of the monitored and controlled physical environment is required to ensure that performance and safety requirements are met. Various tools deal with modeling such multi-domain systems and provide evaluation through simulation. Two such tools — DESTECS and Ptolemy — are examined and compared in this paper, using a case study of an aircraft fuel system. Usability, quantitative, and qualitative comparison criteria are used to give a thorough analysis of the capabilities of the two tools. The contribution of this paper is a description of pros and cons of each tool, helping future users to choose the right tool that suits their needs.

Keywords: embedded systems, modelling, simulation, tool comparison

Financial support: Pierce's work is funded by the European Community's Seventh Framework Program (FP7/2007-2013) under grant agreement no. 248134 (DESTECS) and the UK EPSRC Platform Grant on Trustworthy Ambient Systems. Wolff's work is funded by Terma A/S and the Danish
National Advanced Technology Foundation.

Please cite as: S. Wolff, K. Pierce and P. Derler, 2013. Multi-domain Modelling in DESTECS and Ptolemy - a Tool Comparison. Department of Engineering, Aarhus University. Denmark. 19 pp. - Technical report ECE-TR-15

Cover image: Sune Wolff

ISSN: 2245-2087

MULTI-DOMAIN MODELLING IN DESTECS AND PTOLEMY – A TOOL COMPARISON

Sune Wolff, Department of Engineering,, Aarhus University
Ken Pierce, School of Computing Science, Newcastle University,
Patricia Derler, EECS Department, UC Berkeley

Abstract

Developing embedded systems with high performance and safety requirements is notoriously hard. It is not enough to have a thorough understanding of the control algorithms used, but a deep understanding of the monitored and controlled physical environment is required to ensure that performance and safety requirements are met. Various tools deal with modeling such multi-domain systems and provide evaluation through simulation. Two such tools — DESTECS and Ptolemy — are examined and compared in this paper, using a case study of an aircraft fuel system. Usability, quantitative, and qualitative comparison criteria are used to give a thorough analysis of the capabilities of the two tools. The contribution of this paper is a description of pros and cons of each tool, helping future users to choose the right tool that suits their needs.

Multi-domain Modelling in DESTECS and Ptolemy - a Tool Comparison

Sune Wolff[1], Ken Pierce[2] and Patricia Derler[3]

[1] Department of Engineering, Aarhus University,
Finlandsgade 22, 8200 Aarhus N, Denmark
`swo@iha.dk`

[2] School of Computing Science, Newcastle University,
Newcastle upon Tyne, NE1 7RU, United Kingdom
`kenneth.pierce@newcastle.ac.uk`

[3] EECS Department, 545M Cory Hall, UC Berkeley, CA 94720-1770
`pd@eecs.berkeley.edu`

Abstract. Developing embedded systems with high performance and safety requirements is notoriously hard. It is not enough to have a thorough understanding of the control algorithms used, but a deep understanding of the monitored and controlled physical environment is required to ensure that performance and safety requirements are met. Various tools deal with modeling such multi-domain systems and provide evaluation through simulation. Two such tools — DESTECS and Ptolemy — are examined and compared in this paper, using a case study of an aircraft fuel system. Usability, quantitative, and qualitative comparison criteria are used to give a thorough analysis of the capabilities of the two tools. The contribution of this paper is a description of pros and cons of each tool, helping future users to choose the right tool that suits their needs.

1 Introduction

Embedded systems are characterized by a tight coupling between the computational parts and the physical environment in which the system is operating. The environment is being monitored by the computational system with sensors, while actuators are used to control the environmental parameters. An example is the cruise controller of a car: the velocity of the car, the RPM of the engine as well as the torque applied to the wheels are monitored using various sensors. An embedded controller adjusts actuator output to apply the correct amount of power to the engine and in the case of an automatic transmission applies the correct gear ratio. All this is done to ensure the vehicle reaches the desired velocity in a way that is pleasant to the driver.

Embedded systems often have high performance and safety requirements that must be met. In addition, multiple engineering disciplines are involved in the development of the various parts of the system. Several initiatives in academia advocate the use of a holistic modelling approach to embedded systems development (DESTECS [13], Ptolemy [5,10], Modelica [15] to name a few). The software controller, actuators and sensors, and the physical environment are all modelled within their own domain: software in the *discrete-event* (DE) domain, and the dynamics parts of the system in the *continuous-time* (CT) domain. These different models are then simulated within the same tool (as is the case in Ptolemy and Modelica) or a co-simulation between two tools is carried out (as is the case in DESTECS).

The goal of the work presented in this paper is to analyse the state of the art with respect to available tools by comparing two tools using different approaches to multi-domain modelling and (co-)simulation. In Ptolemy this is done within the same tool, since both DE and CT elements can be described in a single model. In the DESTECS tool-chain the DE elements of a system are modelled in one tool, and the CT elements in another. An extra co-simulation layer performs time synchronisation and transfers variables between the two tools. It is not the intent of this work to declare a winner in the comparison but rather to provide an in-depth investigation of the tools, highlighting the pros and cons of each.

1.1 Ptolemy

Ptolemy II [5,7] is a modelling and simulation framework for heterogeneous systems which is developed at UC Berkeley. The framework is an open-source Java project that has been developed since 1996. Ptolemy Classic, the predecessor of Ptolemy II, was implemented in C++. In this work, we will describe and use Ptolemy II, which is from here on referred to just as Ptolemy.

Ptolemy has been used in industry, for example in HP's DSP Designer and DSP Synthesizer [25] or MLDesigner [22] which use concepts developed in the Ptolemy project. Mirabilis Design [4] have commercialised Ptolemy in the tool VisualSim. Ptolemy has also seen a wide non-commercial use, most noteworthy: the Kepler project [5] and the Building Controls Virtual Test Bed (BCVTB) [6].

The actor-oriented design principle is used to describe systems. Actors are components that communicate via ports. The semantics of an actor and the communication is given by a *Model of Computation* (MoC). Examples for MoCs in Ptolemy are discrete-event (DE), continuous-time (CT), various data flow variants such as synchronous data flow (SDF), process network (PN) and synchronous reactive (SR).

Heterogeneous composition of different MoCs is enabled via hierarchies where every hierarchy level represents exactly one MoC. A special actor, the director, enforces the MoC on each hierarchy level. Composite actors contain actors and, in the case of opaque composite actors, they contain a director. To the enclosing model, a composite actor behaves like an atomic actor with ports for communication. Transparent composite actors do not contain a director and are mere logical groupings of actors.

FMI is an emerging standard –defined by the Modelisar project– which supports both model exchange and co-simulation of dynamic models. Numerous simulation tools already support FMI: AMESim, CosiMate, Dymola, MapleSim, and MATLAB, to mention a few. Ptolemy has a Functional Mock-up Interface (FMI) under development, which will enable co-simulation of Ptolemy models and models created in other tools supporting FMI.

[4] `http://www.mirabilisdesign.com/`
[5] `https://kepler-project.org/`
[6] `https://simulationresearch.lbl.gov/bcvtb`

1.2 DESTECS

DESTECS (Design Support and Tooling for Embedded Control Software) is a consortium with eight members from academia and industry working on collaborative modelling and co-simulation of fault-tolerant embedded systems [3,13]. This is done by combining CT system models with DE controllers in a co-simulation tool-chain.

The DESTECS co-simulation engine acts as a bridge between the CT models made in 20-sim [4] and DE VDM models [11,12] modelled in the Overture tool [19]. The main responsibility of the co-simulation engine is to synchronise time and to pass variables between the two simulation tools. In addition to the tool-chain, a methodology description has been made giving solid guidelines how to create multi-domain models, and specifically how to model faults in the system as well as fault tolerance mechanisms.

Three companies, which are part of the consortium, are working on a case creating a co-model within their own domain. The DESTECS project also has an *industry follow group* consisting of companies interested in the results of the project. Challenges posed by some of these companies have been modelled by the DESTECS project to expand on the successful industrial cases.

1.3 Related Work

In the Columbus project [6] ten different tools and techniques for modelling multi-domain systems are reviewed. The syntax of the individual tools is presented, as well as the different domains they are capable of modelling. A very simple example of a bouncing ball or an electrical RC circuit is modelled to try out the capabilities of each tool. These examples do combine DE and CT elements, but are not good examples of multi-domain embedded systems where a discrete controller interacts with a continuous plant.

Bretenecker et al. are working on an ongoing series of classifications and comparison of simulators for physical modelling. The focus of this work is on the physical modelling capabilities of different simulation tools, but in a recent revision [1] multi-domain modelling and co-simulation have been added to the comparison. The ARGESIM benchmarks [2] (which define 19 different case studies) are used to evaluate more than 20 different simulators. The work done is extensive but focuses on how to solve the different case studies and lacks a more in-depth introduction and comparison of the capabilities of the tool.

In the work of Verma et al. [27], research in software evaluation and comparison is reviewed and a large set of comparison criteria are derived. The tools evaluated are of less interest than the comparison criteria used — we have been inspired by these criteria for the comparison presented here. An important conclusion made by Verma et al. is that no single tool is best for every task. For that reason, the comparison presented here is not an attempt to find the best tool, but rather to evaluate the use of the two tools for solving various tasks.

2 Case Study

The fuel systems on-board aircrafts are complex systems, with multiple tanks in each wing as well as in the tail of the aircraft. Fuel must be transferred between these tanks to ensure the engines always have a sufficient supply of fuel and to maintain the balance of the aircraft front to back as well as sideways. As a means to manage this complexity, computer models have been developed of such systems for the last twenty years (as an early example, see [8]).

In this paper, we use a simplified version of the case presented by Jiminez et al. [17] where only the left side of the aircraft fuel system is modelled and the outermost tank in the wing is removed. An overview of the fuel tanks in our study and how they are connected can be seen in Fig. 1.

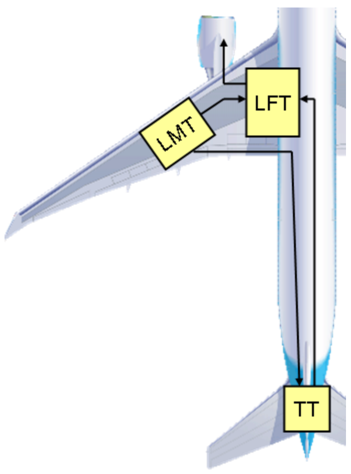


Fig. 1. Overview of the the fuel tanks in the case study

There are three tanks in our case study: the feeder tank (LFT) supplying the engine with fuel, the middle tank (LMT) carrying extra fuel and the trimmer tank (TT) placed in the rear end of the aircraft, which is used to balance ("trim") the aircraft. At takeoff, TT must be empty for safety reasons, but once cruise altitude has been reached TT must be filled to obtain better balance of the aircraft. The LFT must always have sufficient supply of fuel to the engine which continuously consumes fuel (more during takeoff than at cruise altitude). If the fuel level of the feeder tank reaches a minimum threshold, fuel must be pumped from the LMT to the feeder tank. If LMT is empty, fuel must be transferred from TT to ensure a sufficient level of fuel for the engine. Finally, before landing the aircraft TT must be emptied for safety reasons.

Two scenarios will be used in the comparative study:

"Sunshine" scenario: The aircraft takes off from ground level and when cruise altitude is reached, the ascend is stopped. The aircraft stays at this altitude until the

LFT reaches the low threshold at which point the aircraft starts descending. This scenario tests the normative behaviour of the fuel system.

“Faulty” scenario: In this scenario the fuel level sensor of the TT malfunctions and no signal is sent to the controller. This scenario tests that the controller can handle the malfunctioning sensor, while maintaining a sufficient supply of fuel to the engine ensuring a safe landing of the aircraft.

2.1 Modelling in Ptolemy

Airplane fuel system models in Ptolemy are studied in [9]. The various models highlight challenges in modeling cyber-physical systems. In this paper, we extend the fuel system model to a more realistic version which is presented in Fig. 2.

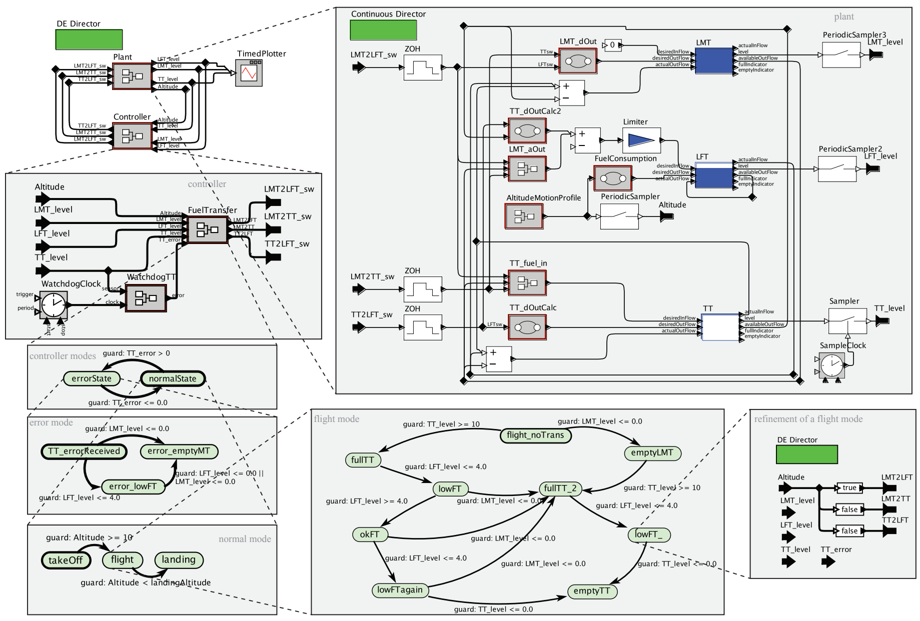


Fig. 2. The fuel system model in Ptolemy.

The top level model consists of two composite actors: a controller and a plant. As the top level MoC, DE was chosen. The plant is modeled as a CT component, and the controller inherits the DE MoC form the top level. Inputs to the plant model are discrete and have to be converted to continuous signals with zero-order-hold components. Outputs are discretized with samplers. The plant comprises the three tanks as described above; fuel mover components that transfer fuel between the tanks; and an altitude profile. The control logic is specified via modal models and mode refinements define the control outputs. Modal models are used to express:

(a) Normal and faulty operation. The fault scenario describes a fault in the TT. Therefore, the signal from the TT (TT_error) is checked and based on the value of this input, the mode is switched. Refinements of the states express the different behaviors.
(b) The flight modes takeoff, flight and landing.
(c) Various error modes.
(d) Various modes that, based on certain fuel levels in the tanks, move fuel between the tanks. This represents the main control logic.

Fig. 3 shows the outputs of the simulation. The fuel levels of the three tanks as well as the altitude of the aircraft are monitored.

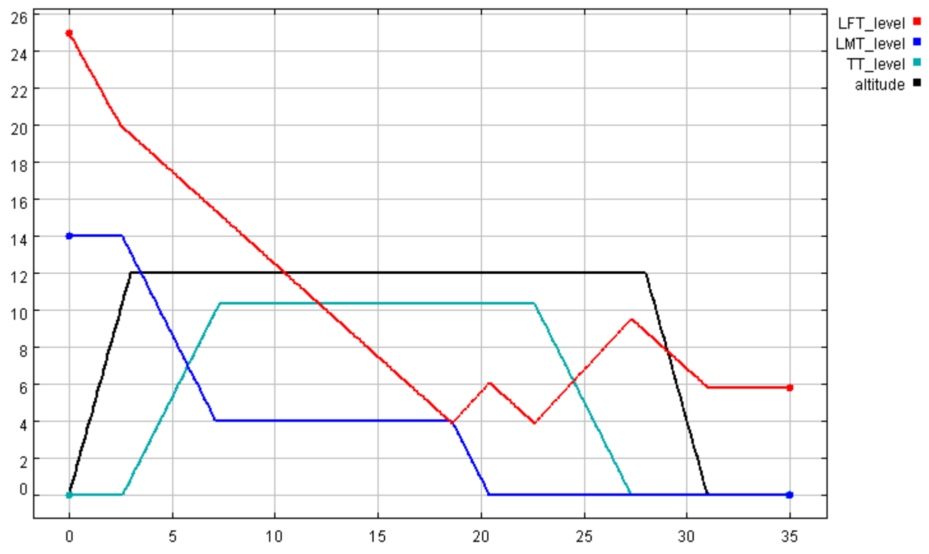


Fig. 3. Fuel levels obtained through simulation in Ptolemy.

2.2 Modelling in DESTECS

The three fuel tanks –as well as the dynamics of transferring fuel between these– were modelled as a CT model in the 20-sim tool, while the discrete control scheme was added as a VDM-RT model in the Overture tool. 20-sim includes pre-defined iconic blocks for modelling components in the hydraulic domain which could have been used in order to add realism to the model with regards to the flow of fuel: volume components for the tanks; pump components; and hydraulic inertia in the pipes connecting the fuel tanks. To better compare the two tools it was chosen to model the tanks using an equational approach similarly to what was done in the Ptolemy tool. Fig. 4 gives an overview of the DESTECS fuel system model in the 20-sim editor.

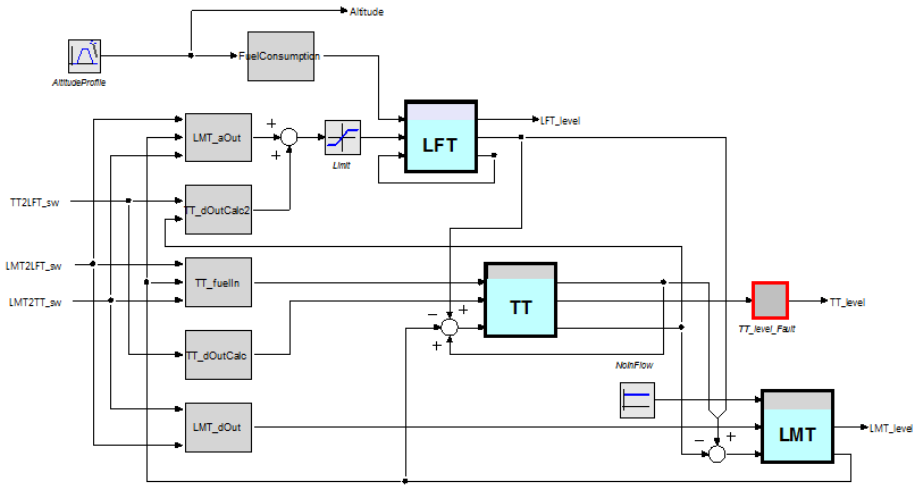


Fig. 4. The fuel system in 20-sim.

VDM-RT is an object-oriented language enabling the modeller to express the DE controller using inheritance, polymorphism and composition. The faulty sensor was modelled as a concrete subclass of an abstract sensor class. To run the faulty scenario, the faulty subclass was instantiated for the TT fuel level sensor and the normal sensor subclass for the remaining sensors of the system. To trigger the faulty sensor the DESTECS Command Language (DCL) was used. DCL is a scripting language created in order to simulate user input and activate latent fault behaviours during a co-simulation. Listing 1 shows the simple DCL script used to activate the faulty TT fuel level sensor after 3.9 seconds of simulated time.

```
when time >= 3.9 do
(
    ct boolean TTLEVELERROR := true;
);
```

Listing 1. DCL script activating the faulty trimmer tank level sensor.

To setup a simulation run a *Launch Configuration* is created inside the DESTECS tool, defining the entry point of the simulation, simulation time, type of ODE solver, etc. Separate launch configurations were created for the normal and faulty scenario ensuring that the correct concrete sensor class implementations were used and that the DCL script was activated for the faulty scenario.

To visualise the complete system model the 3D animator built into 20-sim was used. This enables the model designer to use parameters of the model to scale, rotate or move simple objects like cubes, cylinders or planes. In addition, objects can be imported from other 3D modelling tools in a variety of standard formats. To add to the visual presentation an aircraft model was imported, and simple cubes were added as the fuel tanks - a screenshot from the resulting 3D animation can be seen in Fig. 5.

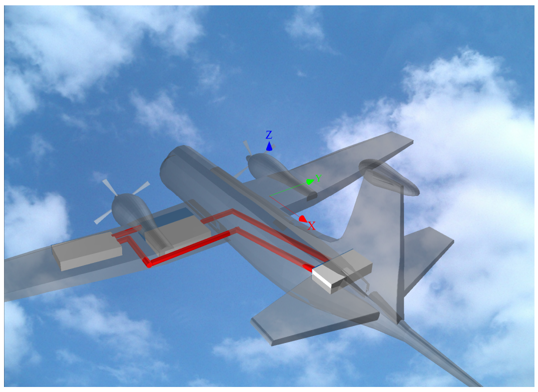


Fig. 5. Screenshot of the 3D animation of the DESTECS fuel system model.

3 Comparison Criteria

The comparison criteria used during the comparison of the two tools are divided into three parts: usability criteria giving an overview of the accessibility of the tools, quantitative criteria comparing the tools on simulation speed and similar measurable metrics, and finally qualitative criteria giving an overview of the pros and cons of each tool.

3.1 Usability Comparison Criteria

This part of the comparison focuses on the usability of the two tools: how easy is it for a new user to install the tool and get initial help when technical issues are encountered? The usability criteria are divided into the following sub-categories:

Installation: This category covers the installation experience of the tools. The following gives examples of questions that are answered in this category: How easy was it to install the tool? Do the tools depend on any additional software (Java runtime environment or similar) to be installed in order to work, and are these installed automatically as part of the installation process? Are there any license required to use the tool, and how easy is it to acquire such a license?

Updates: This category covers anything related to updating the tools. How frequent are public updates made available? How are users notified of these updates? Is it possible to use developer builds? If yes, how easy is it to access these and what is the frequency of release?

Model management: This category describes how easy it is to manage the models created in the tools: How many files are created for a single model, and how are they organised? Do the tools support the use of different alternative implementations of submodels, and how easy is it to switch between these?

Extensions: How easy is it to extend the tool with additional capabilities? This category analyses internal extensions with the addition of new functional blocks for example, as well as external extensions like linking to other tools or export of data to be used by other tools.

Help: This category answers questions like: How easy is it to get help concerning technical issues? Are there existing examples to get inspiration from? Is there an active community with a forum or mailing list to ask technical questions? How well is the tool documented with regards to user guidance? Is there an in-tool help function to quickly access the documentation of the tool?

3.2 Quantitative Metrics

This part of the comparison focuses on quantitative comparison criteria where the two tools can be benchmarked against each other, resulting in metrics that are directly comparable.

The (co-)simulation speed, time to load the model, and memory consumption are all metrics which are compared. In addition, the number of *Ordinary Differential Equation* (ODE) solvers each tool support is compared, divided into fixed-step and variable-step solvers. For the quantitative comparison, a laptop with the following specifications was used:

Laptop	Lenovo ThinkPad® T500
CPU	Intel® Core^{TM}2Duo T9400 (2.53GHz, 6MB L2, 1066MHz FSB)
Operating system	Windows® 7 Professional 64-bit
Installed memory (RAM)	8 GB
Graphical Processing Unit	ATI Mobility Radeon^{TM} HD 3650

Table 1. Technical specifications of the PC used for the comparative tests.

3.3 Qualitative Comparison Criteria

In the qualitative category the following comparison criteria were chosen prior to creating the two models:

Discrete-event controller expressiveness: Are there limitations as to what can be modelled in the DE controllers in the two tools? Is it possible to model an object-oriented architecture? Do the tools support the use of abstract data types?

Hierarchical modelling: Do the tools support a hierarchical structure in the models?

Model reuse: How easy is it to reuse parts of the model? Can submodels be created and linked to other projects, or is model reuse only supported by copy-paste? Do updates to submodels automatically mitigate to projects in which they are used?

Fault modelling: How easy is it to model faults and fault-tolerance mechanisms? Do the tools contain implicit help to this or must it be done explicitly by the model designer?

Extensions of the model: How easy is it to extend models, for instance adding additional tanks?

In addition to these comparison criteria defined prior to the comparative work, additional criteria were discovered during the creation of the two models. If at any time the model designers encountered some issue or something well supported by one of the tools, this feature (or lack thereof) was added to the comparison criteria. These are described under results in Section 4.4.

4 Comparison Results

The most recent public releases of the tools were used for all the tests done. For Ptolemy version 8.0.1 from October 28th 2010[7] and for DESTECS version 1.3.3 from April 2012. At time of writing the DESTECS project is still producing monthly releases of the tool — please visit the download page[8] to get the most resent version of the tool.

4.1 Usability — Ptolemy

Installation: Ptolemy is an open-source framework built in Java and it is released under the BSD license [23]. Installer and guides for installing on Windows, Linux and OSX are available online and maintained regularly. Ptolemy comes with various packages where some are by default disabled and not compiled. In order to compile these packages, additional libraries might be necessary. Ptolemy can be executed as a standalone application or embedded into the Eclipse development framework.

Updates: Official updates of the tool are released on an annual or biennial cycle. It is also possible to work with the latest build of the source which is available via a subversion repository. The code base for the project (as of beginning of 2012) takes about 1GB of hard disc space. An installation from the source code repository takes up to one hour and following the guides is a lengthy process though well documented. Installers are also built nightly to ease this process.

Model Management: Ptolemy models are stored as XML files in a syntax called MoML [21]. A model can be stored as one or multiple XML files. Ptolemy includes a special mechanism for object-oriented modelling. A class can be implemented and instances of that class can be reused in different parts of the model. The object-oriented design principles in Ptolemy are unique [20]. Different versions of the model have to be stored in different files. There is no automatic support for version control.

[7] http://ptolemy.eecs.berkeley.edu/ptolemyII/ptII8.0/index.htm
[8] http://www.destecs.org/downloads.html

Extensions: Ptolemy comes with a library of directors and actors. Compared to commercial tools such as Matlab/Simulink, this library is fairly small. Extending Ptolemy with new functionality such as models of computation or new actors means creating a new Java class which implements specific interfaces. Tutorials on how to implement a new actor or a new MoC are available online.

Help: The Ptolemy project is released with many demos that illustrate how to create models with different MoCs. Help is also provided via various mailing lists such as the Ptolemy interest mailing list or the Ptolemy hackers mailing list. The code base is constantly changing as a number of students, researchers and industrial collaborators are extending and experimenting with the tools. All extensions to Ptolemy are checked to ensure they still conform to the current code base. A convenient way to do this is by creating regression tests. Such tests can be written in TCL or implemented in the model by using a special test actor. This actor *learns* the correct values in a *training run* and then checks subsequent executions of the model against these values. Code that is submitted to the source-tree undergoes strict checks and has to adhere to coding guidelines which are documented. Documentation is fairly extensive and utilizes Javadoc [24] to automatically generate documentation.

4.2 Usability — DESTECS

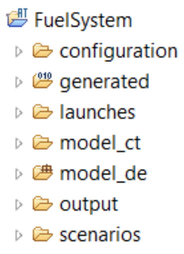
Installation: The DESTECS tool is an open source tool developed on top of the Eclipse [16] *Integrated Development Environment* (IDE)and official releases can be downloaded from SourceForge. In addition, nightly builds can be accessed directly from the build server[9]. A part of the installer is the open source IDE Overture. A free *viewer* version of the commercial tool 20-sim is also installed which gives access to all of 20-sim's capabilities except saving models. To enable this last feature the free license must be upgraded to a full professional license. 20-sim is only built for Windows (XP and newer versions) so the DESTECS tool only runs on this platform even though Overture is built for Windows, OSX as well as Linux.

Updates: Only a few official updates of the DESTECS toolchain have been released on an annual cycle. Developer builds used internally in the DESTECS project have been released on a monthly basis. As of October 2012 public releases outside the consortium have been made available on the project download page.

Model Management: Different formats are used for the different parts of a DESTECS co-model: 20-sim models are stored in the XML-based proprietary format *emx*, VDM models are stored in the VDM-RT format and co-simulation settings are stored as launch configurations in the Eclipse based IDE Overture.

Fig. 6 gives an overview of the file hierarchy inside the DESTECS tool. The *configuration* folder holds the contract file which defines the interface between the DE and CT models as well as the link file which links the monitored and controlled variables

[9] `http://build.destecs.org/`



FuelSystem
configuration
generated
launches
model_ct
model_de
output
scenarios

Fig. 6. File hierarchy in the DESTECS tool.

from the contract to the VDM model. The *launch* folder holds the launch configuration defining all the co-simulation settings. The *model_ct* folder holds the 20-sim model and the *model_de* folder holds all the classes of the VDM controller. Code coverage of the VDM controller and log files are generated in the *output* folder. In the *scenarios* folder all DCL scripted scenarios are stored which are used to introduce errors into the model or to model user interaction.

20-sim does not support an object-oriented structure of the CT models like Ptolemy does. Instead, sub-models can be created and imported into other models (multiple instances if needed). If changes are needed to this part of the model, only the separate sub-model needs to be changed and the imported instances can be updated to reflect the changes. There is no automatic support for version control inside the DESTECS tool, but since it is based on Eclipse, third-part plug-ins like Subclipse [10] can be used for this task inside the IDE.

Extensions: Since the Overture tool is open-source it is possible for the public to make extensions on the DE side of the DESTECS toolchain. 20-sim is a commercial tool, so it is not possible to make extensions to the CT simulation tool. It is possible, though, to define new blocks being either equation or graphical based. This enables the model designer to manually describe the differential equations of the CT components or to create graphical representations using the underlying Bond graph [18] technology.

Help: 20-sim comes with a large library of example projects, exemplifying the use of: 1D, 2D and 3D mechanics modelling, block diagrams, Bond graphs, electric motors, hydraulics, signal processing, and many more. Numerous sample VDM-RT projects can be downloaded from the project wiki page [11] to get a good introduction to DE modelling in the Overture tool. Currently only 15 publicly available co-model examples exist for the DESTECS tool — these are small example projects explaining the use of all of the capabilities of the tool. It is planned to expand on this in future years.

[10] `http://subclipse.tigris.org/`
[11] `http://overturetool.org/?q=node/15`

4.3 Quantitative

Clean installations of both tools were made in order to gather data on the installation process. The startup of the tools were monitored as well as the duration of running simulations of the two scenarios. Finally, consumption of system resources was monitored. Results of the quantitative comparison can be seen in Table 2.

Criteria	DESTECS	Ptolemy	Unit
Installer size	147599	229022	KB
Installation duration	0:59	2:53	min
Size of installed tool	264	229	MB
Time to startup tool	4.3	3.1	sec
Time to load model	*	3.7	sec
Size of model	1098 (167)	672	KB
Simulation speed			
- Scenario 1	7.3	6.1	sec
- Scenario 2	18.3	5.1	sec
Memory consumption			
- Idle (model loaded)	167540	212740	KB
- Peak (during simulation)	260640	580284	KB
CPU consumption			
- Idle (model loaded)	0	0	%
- Peak (during simulation)	94	99	%
# ODE solvers			
- Fixed step	5	2	
- Variable step	5	2	

Table 2. Overview of the qualitative comparison results.

The Ptolemy installer takes a lot longer to run than the DESTECS equivalent, but most of the installation time was spent on installing full source code of Ptolemy. If the source code is not needed, installation time as well as the final size of the tool are decreased.

Since DESTECS is built on top of the Eclipse platform, models are as such not loaded into the tool. Instead they exist in a workspace which is automatically loaded as part of the tool — hence no measurement of model load duration was possible for the DESTECS tool.

As described in Section 1.2, a 3D animation was included in the DESTECS model. The core objects of the aircraft model was created in an external 3D modelling tool called Blender [12] and imported into the DESTECS model. These objects were 931KB in total, so to compare the model size directly with Ptolemy (Ptolemy has a Java3D

[12] `http://www.blender.org/`

interface which was not used in the work presented here) these objects should be subtracted, resulting in a DESTECS model size of only 167KB.

In the DESTECS tool, simulation of the faulty scenario was three times slower as the "sunshine" scenario. In order to trigger the error an additional value had to be passed between the Overture and 20-sim tools of DESTECS. The co-simulation engine is only optimised with regards to the passing of values defined in the contract between the two tools, and not these additional values triggered via the DCL script. This is the reason for the significant performance decrease in the simulation of the faulty scenario in the DESTECS tool.

The Ptolemy tool performed better in the faulty scenario. This could be due to the fact that when the TT fuel level sensor malfunctions, Ptolemy stops drawing the fuel-level in the 2D graph. This indicates that rendering the graphs is quite demanding in Ptolemy, so reducing the plotting will increase performance of the tool.

When running multiple subsequent simulations the peak memory consumption of Ptolemy slowly increased until settling at the value listed in Table 2. This indicates that the tool has some memory of state between simulations. If it had been a case of a memory leak, the memory consumption would keep increasing — this was not the case.

Since 20-sim is a fully fledged CT modelling tool, it has more diversity in the choice of ODE solvers. The wider choice can help optimising the duration of simulations since it will be easier to find one that is suitable for the model at hand. Optimisations like this requires a deep insight into the CT model though, so they are reserved for experts in this area.

4.4 Qualitative

This subsection describe several specific benefits and limitations of the two tools which were discovered during creation of the models.

DE controllers: Actor-oriented programming in Ptolemy enables the modeller to create hierarchical models where each level in the hierarchy potentially can have a different model of computation. The DE controller was modelled as a hierarchically layered modal model — the top layer switching between normal and faulty mode, and individual underlying modal models managing the transfer of fuel between the different tanks in each of these two modes. The hierarchical structure is achieved by making new modal models as refinements of the individual modes in a modal model higher in the hierarchy as can be seen in Fig. 2. This has the unfortunate side effect that the controller is 7 levels deep and has 15 individual modal models (even without counting all the individual state refinements which sets the value states) which makes it complex to navigate.

The use of a modal model description of the controller also have some major advantages though. In embedded software the controller is commonly structured as a state machine since the controller has state dependent behaviour. Using the same notation in the model makes it easier for embedded software engineers to learn how to use the tool without having to learn all the syntactic subtleties of a new language.

In the DESTECS tool the model designer has a full object-oriented language in which to express the DE controller. This enables the use of class inheritance, composition, polymorphy and similar object-oriented structures. This is a major advantage when describing more complex controllers where state machines can become too big to manage. The object-oriented structure also enables the modeller to try out different control strategies by having an abstract controller main class with several alternative concrete controller implementations each specifying different control schemes. This approach was used when modelling the faulty sensor which was defined as a concrete subclass of an abstract sensor class. To run the faulty scenario, the faulty subclass was instantiated for the TT fuel level sensor and the normal sensor subclass for the remaining sensors of the system. This is a great way of using alternative versions of the model in different scenarios without polluting the model with unneeded information and without forcing manual alterations to the model.

CT plant: The CT plant model in Ptolemy was created using a signal-based block diagram. Some pre-defined CT actors like integrators and various sources and sinks are available, but more intricate actors must be manually created using either custom expressions or modal models. When modelling the altitude profile for the aircraft an expression actor was used, manually ramping the output from zero to cruise altitude; stay at cruise altitude for a certain period of time; and finally decent to zero altitude for landing the aircraft. This was done using three nested if-then-else constructs written using the ternary operator, since the expression actor only allows single line expressions. This makes the expression quite hard to read and an unfortunate source of errors. In DESTECS the CT model was created in the 20-sim tool. To model the same altitude profile the built-in *motion profile wizard* was used — here the user is guided through a series of steps constructing the desired signal. This is done by making a single or continuous combination of a series of ramp, constants, (co)sine or similar types of sources. The user gets a graphical overview of the generated output as a visual validation that the correct motion profile has been generated. This is a much more user-friendly approach to creating custom made sources compared to the Ptolemy equivalent.

In addition to blocks from pre-defined libraries, the user may also define their own blocks. These blocks may contain equations, Bond graphs, or further graphical models. This allows the user to structure their models hierarchically. Users may also add their own blocks to the library for later reuse. A block may have more than one "implementation" defined allowing alternative behaviours to be modelled, such as faults. The DESTECS tool allows the user to select a specific implementation before co-simulation.

The simpler (so-called *signal level*) model of the fuel tanks was built in 20-sim. A key reason for this was to permit a better comparison with Ptolemy in terms of functionality of the CT model. This choice means that if a tank is full the back pressure must be modelled explicitly between each pair of tanks. The Bond graph notation supported by 20-sim could be used here, which offers a more compositional solution. All connections are "two way" so that the back pressure from a full tank is automatically taken into account. Further tanks can simply be connected and the causality determined runtime. Qian has produced an initial Bond graph model of the fuel tanks [26]. This model includes more realistic tanks, pipes and pumps, with the flows of fuel appearing as curves on the plots. The model also includes a calculation for changing the centre of gravity during the flight (along the nose-tail axis of the plane only), bringing it closer to the original model by Jiminez et al. [17]. Note that it was possible to plug this alternative CT model directly into the existing co-model, permitting the use of the same DE controller and test scenarios.

Fault Modelling: In the Ptolemy model, the TT fuel level sensor is monitored using a *watchdog*. If two consecutive readings are missing an error is issued, which switches the DE controller to its error state. Ptolemy includes an alternative implementation of fault handling [14]. This mechanism allows for models to throw errors in the same manner as programming languages such as Java do. These errors are populated up the hierarchy until there is a model that can deal with the error. A special error transition in a modal model catches the error and the model switches to a fault mode. The cited paper mentions how to deal with timing errors in such models but this error handling mechanism could be extended to arbitrary errors like the one in the case study presented here.

One of the main goals for the DESTECS project when modelling realistic and faulty behaviour is to avoid what might crudely be termed *pollution* of models with descriptions of these non-ideal behaviours. Hence, it is advocated to keep a clear distinction between ideal and realistic/faulty behaviours in models. One way of achieving this is by using the DCL notation as described in Section 1.2. Using a combination of sub-classing to model both normal and faulty sensors and using the DCL script to trigger the error is a great way of ensuring that the (de)activation of the faulty sensor is decoupled from the rest of the model without polluting the model.

Visualisation of the Simulations: The 2D plotters of Ptolemy are simple and user friendly. To add a signal to a plotter a connection is simply drawn from the signal to the plotter. When comparing variable values which exist at different hierarchical levels of a model, the signals cannot seamlessly be dragged to the plotter. To plot two such variables in the same window they must be passed in the interface of the hierarchical blocks. Alternatively, the MATLAB interface of Ptolemy can be used, passing the relevant variables and plotting these in MATLAB. In the 2D graphs of 20-sim the user chooses any variable from the model which is then plotted, which makes it easier to compare variable values existing in different parts of the model.

20-sim includes a 2D graph editor enabling the use to pick any signal or variable value in the entire model from a hierarchical tree structure, add a name and color and have them plotted either collided or in separate graphs in the same window. This makes it easier to compare signals existing in different parts of the model. In addition an interface to the tool Octave [13] has been implemented in DESTECS enabling the user to choose any signal from the CT or DE model to be plottet in a 2D graph. This gives the user further possibilities to monitor and compare variable values in the model.

As mentioned in Section 1.2 20-sim also includes a 3D animator which can be used to give an even better representation of the system model. Such 3D animations have

[13] `http://www.gnu.org/software/octave/`

been used with great success to explain the functionality of co-models in the DESTECS project. Ptolemy has a Java3D interface which can be used with similar results.

5 Concluding Remarks

As stated at the beginning of this paper, the intent of the comparison was not to declare a winner between the two tools, but rather to offer the reader insight into their use on a common case study, presenting information that can help users make an informed choice about which tool to use. Both tools were able to model and simulate the case study, including modelling the plant in the CT domain and the controller in the DE domain. Both tools were able to model a single fault in a sensor with associated fault tolerance in the controller.

There are a few key ways in which the work on this paper could be expanded to give a better comparison. First, the case study was deliberately kept simple, however this does not necessarily exercise both tools to their full potential. The current case study could be expanded to include more features such as more tanks, more faults and fault tolerance. Especially the CT elements of the case study are simplified — fuel moves instantly between tanks; fuel transfer is not affected by gravity; and the movement and rotation of the aircraft is completely abstracted away. Adding some of these element would ideally push both tools towards their limits and demonstrate if one tool is better than the other *for certain situations*. For example if the fault-tolerant controller behaviour became really complex, the object-oriented abilities of DESTECS might show that tool to be the best choice. Another way to compare the tools is to try other models from different domains. This might offer a way for users to decide between the tools based on the domain of interest.

Finally, an extension of this comparison to other tools would be of great interest and offer insight into a broader range of modelling and (co-)simulation capabilities. The definition of challenge problems together with a set of evaluation criteria for tools would allow for better comparison and evaluation for specific needs of the tool users. The comparison criteria and case study defined in this paper, could be used as a starting point for creating such a common platform for collaborative modelling and (co-)simulation tools.

Acknowledgments

The authors would like to thank Peter Gorm Larsen and Christian Klein, from the DESTECS consortium, and Stavros Tripakis and Christopher Brooks, from the Ptolemy project, for valuable comments on earlier versions of this paper. Pierce's work is funded by the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 248134 (DESTECS) and the UK EPSRC Platform Grant on Trustworthy Ambient Systems. Wolff's work is funded by Terma A/S and the Danish Agency for Science, Technology and Innovation.

References

1. Breitenecker, F., Popper, N., Zauner, G., Landsiedl, M., Rler, M., Heinzl, B., Krner, A.: Simulators For Physical Modelling - Classification and Comparison of Features /Revision 2010. In: Snorek, M., Buk, Z., Cepek, M., Drchal, J. (eds.) Proceedings of EUROSIM 2010 - 7th Congress on Modelling and Simulation. vol. 2, pp. 1051–1061 (September 2010)
2. Breitenecker, F., Wassertheurer, S., Popper, N., Zauner, G.: Benchmarking of simulation systems–the argesim comparisons. In: Proceedings of the First Asia International Conference on Modelling & Simulation. pp. 568–573. AMS '07, IEEE Computer Society, Washington, DC, USA (2007), http://dx.doi.org/10.1109/AMS.2007.22
3. Broenink, J.F., Larsen, P.G., Verhoef, M., Kleijn, C., Jovanovic, D., Pierce, K., F., W.: Design support and tooling for dependable embedded control software. In: Proceedings of Serene 2010 International Workshop on Software Engineering for Resilient Systems. ACM (April 2010)
4. Broenink, J.F.: Modelling, Simulation and Analysis with 20-Sim. Journal A Special Issue CACSD 38(3), 22–25 (1997)
5. Buck, J.T., Ha, S., Lee, E.A., Messerschmitt, D.G.: Ptolemy: A Framework for Simulating and Prototyping Heterogeneous System. In: Int. Journal of Computer Simulation (1994)
6. Carloni, L., Benedetto, M.D.D., Pinot, A., Sangiovanni-Vincentelli, A.: Modeling techniques, programming languages, design toolsets and interchange formats for hybrid systems. Deliverable DHS3, Project IST-2001-38314 COLUMBUS project - Design of Embedded Controllers for Safety Critical Systems (March 2004)
7. Davis, J., Galicia, R., Goel, M., Hylands, C., Lee, E., Liu, J., Liu, X., Muliadi, L., Neuendorffer, S., Reekie, J., Smyth, N., Tsay, J., Xiong, Y.: Ptolemy-II: Heterogeneous concurrent modeling and design in Java. Technical Memorandum UCB/ERL No. M99/40, University of California at Berkeley (July 1999)
8. Davis, T.: Aircraft fuel system simulation. In: Aerospace and Electronics Conference, 1990. NAECON 1990., Proceedings of the IEEE 1990 National. pp. 905 –911 vol.2 (may 1990)
9. Derler, P., Lee, E.A., Sangiovanni-Vincentelli, A.: Modeling cyber-physical systems. Proceedings of the IEEE special issue on CPS (December 2011), http://chess.eecs.berkeley.edu/pubs/843.html
10. Eker, J., Janneck, J., Lee, E., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, S., Xiong, Y.: Taming heterogeneity – the Ptolemy approach. Proc. of the IEEE 91(1), 127–144 (January 2003)
11. Fitzgerald, J.S., Larsen, P.G., Verhoef, M.: Vienna Development Method. Wiley Encyclopedia of Computer Science and Engineering (2008), edited by Benjamin Wah, John Wiley & Sons, Inc.
12. Fitzgerald, J., Larsen, P.G.: Modelling Systems – Practical Tools and Techniques in Software Development. Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, UK, Second edn. (2009), ISBN 0-521-62348-0
13. Fitzgerald, J., Larsen, P.G., Pierce, K., Verhoef, M., Wolff, S.: Collaborative Modelling and Co-simulation in the Development of Dependable Embedded Systems. In: Méry, D., Merz, S. (eds.) IFM 2010, Integrated Formal Methods. Lecture Notes in Computer Science, vol. 6396, pp. 12–26. Springer-Verlag (October 2010)
14. Forbes, S.S.: Toward an error handling mechanism for timing errors with java pathfinder and ptolemy ii. Tech. Rep. UCB/EECS-2010-123, EECS Department, University of California, Berkeley (Sep 2010), http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-123.html
15. Fritzson, P., Engelson, V.: Modelica - a unified object-oriented language for system modelling and simulation. In: ECCOP '98: Proceedings of the 12th European Conference

on Object-Oriented Programming. pp. 67–90. Springer-Verlag (1998), http://www.modelica.org/documents/ModelicaSpec32.pdf
16. Ian Sommerville, Ray Welland, S.P., Smart, J.: The ECLIPSE User Interface. Software - Practice and Experience 19(4), 371–391 (April 1989)
17. Jimenez, J.F., Giron-Sierra, J.M., Insaurralde, C., Seminario, M.: A simulation of aircraft fuel management system. Simulation Modelling Practice and Theory 15(5), 544 – 564 (2007)
18. Karnopp, D., Rosenberg, R.: Analysis and simulation of multiport systems: the bond graph approach to physical system dynamic. MIT Press, Cambridge, MA, USA (1968)
19. Larsen, P.G., Battle, N., Ferreira, M., Fitzgerald, J., Lausdahl, K., Verhoef, M.: The Overture Initiative – Integrating Tools for VDM. ACM Software Engineering Notes 35(1) (January 2010)
20. Lee, E.A., Liu, X., Neuendorffer, S.: Classes and inheritance in actor-oriented design. ACM Transactions on Embedded Computing Systems (TECS) 8(4), 29:1–29:26 (2009), http://ptolemy.eecs.berkeley.edu/publications/papers/07/classesandInheritance/index.htm
21. Lee, E.A., Neuendorffer, S.: Moml - a modeling markup language in xml - version 0.4 (2000)
22. MLDesign Technologies: Mldesigner (2012), http://www.mldesigner.com
23. Open Source Initiative: Open source initiative osi - the bsd license:licensing (2012), http://www.opensource.org/licenses/bsd-license.php
24. Oracle: Javadoc tool (2012), http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html
25. Pino, J.L., Ha, S., Lee, E.A., Buck, J.T.: Software synthesis for dsp using ptolemy. Journal of VLSI Signal Processing 9, 7–21 (1993)
26. Qian, J.: Co-Design of Embedded Systems: an Aircraft Fuel Tank (to appear). Master's thesis, Newcastle University, UK (September 2012)
27. Rajesh Verma, A.G., Singh, K.: A Critical Evaluation and Comparison of Four Manufacturing Simulation Softwares. Kathmandu University Journal of Science, Eengineering and Technology 5(1), 104–120 (January 2009)

Department of Engineering
Aarhus University
Edison, Finlandsgade 22
8200 Aarhus N
Denmark

Tel.: +45 4189 3000