Co-Design Tool Construction Using APICES

Ansgar Bredenfeld
GMD
Institute for Autonomous Intelligent Systems (AiS)
D-53754 Sankt Augustin, Germany
+49-2241-14-2841
bredenfeld@gmd.de

ABSTRACT

In this paper, we present our approach to automate the development process of co-design tools. We demonstrate with a non-trivial real world example how we can accelerate the tool design process using the software prototyping environment APICES.

In a very short time we constructed the tool Dual Dynamics Designer (DDD) which supports a novel methodology in robot software development. DDD allows to edit a complex differential equation-based specification of dynamic robot behavior via an intuitive graphical interface and automatically generates microcontroller code in C as well as a simulation model in Java from it.

Speed-up of the tool design process is primarily achieved by a rigorous top-down tool modeling approach in combination with a highly configurable tool frame generator.

1. INTRODUCTION

In many cases, co-design tasks can be tackled with design systems that consists of a set of available hardware and software tools [2][7][8][12][15]. This holds especially true for co-design problems that can be partitioned into more or less independent sub-problems. On the other hand, we face problems that require highly specialized environments which are composed of special tools tailored to solve design problems occurring in small classes of target systems. In these environments, available tools are at best building blocks that solve standardized sub-problems, e.g. the simulation or synthesis of separable sub-systems. The rest are special tools which have to be designed.

The resulting problem is a significant effort in terms of time and money needed for the development of tools and integration environments. Often the development of new tools is avoided due to high costs and long development time, although these tools could speed-up steps in a co-design process. This situation is typical for innovative development departments and research institutes.

In this paper, we present a solution approach to automate the design of specialized niche tools. We demonstrate that the UML-based software prototyping environment APICES [3][4] allows to design co-design tools within a very short time.

As example, we construct an editor/generator tool for robot software. The tool is based on Dual Dynamics [9] - a novel, mathematical design scheme for complex robot control systems. Such robot control systems typically consist of many (hundreds) differential equations, which are approximated on digital robot hardware by difference equations. The Dual Dynamics design scheme provides a set of mathematical constraints which help the designer to define a transparently structured ensemble of differential equations. However, the language of Dual Dynamics is primarily mathematical. In order to put this approach into practice, we needed a new tool which translates the mathematical constraints into practical, concrete design rules, and which supports the generation of executable C code from mathematical formulae.

We developed a tool called Dual Dynamics Designer (DDD). It is used for design, monitoring and test of an autonomous robot platform [11] which is supplied with various sensors (camera, gyro, IR-distance sensors, bumpers, laser scanner) and actuators (motors, camera servo, distance sensor servo). We have to develop the tool in a very short time, since we intend to participate in the international RoboCup contest [10] in summer 1999. We consider this project setting typical for many other time critical co-design tool developments.

This paper is structured as follows. Section 2 gives an introduction to the tool we have to construct in order to solve our design problem. Section 3 sketches the software prototyping environment APICES. Section 4 describes the design of the DDD tool with APICES. Section 5 summarizes some details on the design process and the DDD tool. Section 6 briefly compares our work to related work before we end the paper with conclusions and an outlook to next steps.

2. DUAL DYNAMICS DESIGNER (DDD)

Dual Dynamics Designer (DDD) is a specialized tool to generate robot control software from a mathematical specification of robot behaviors. In this section, we give a brief overview of the principles behind Dual Dynamics in order to elucidate the functionality of the tool.

In Dual Dynamics, a robot behavior control system is specified through a considerable number of ordinary differential equations (ODE's). They fall into two major categories, namely, sensor pre-processing and behavior control. Within each of these categories, further structuring constraints prescribe a grouping of ODE's into subsystems with specific interaction mechanisms between them. Figure 1 sketches the overall architecture of the control system.

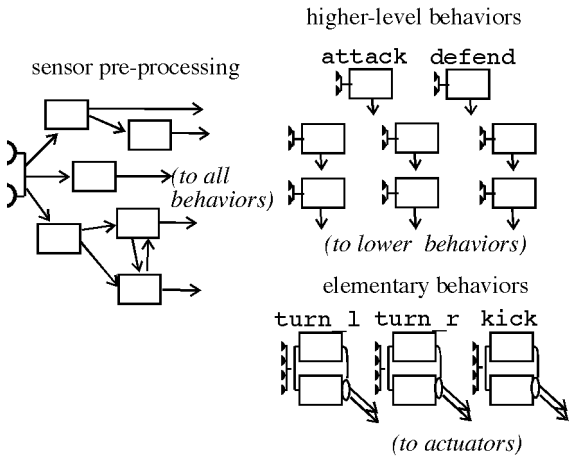

Figure 1. A Dual Dynamics control architecture. Boxes correspond to subsystems made from one to many ODE's.

Sensor pre-processing is organized as a network of sensor filters. Each sensor filter may process raw sensor outputs or intermediate sensor variables produced by other sensor filters. Importantly, cyclic connections are admissible.

Behavior control is organized in terms of a hierarchy of so-called "behaviors". In the mobile robotics community, behaviors [6] are the central building block of robot control systems. They can range from short-term, stereotyped "reflexes" like "kick ball" to long-term, flexible activity patterns like "defend". In the Dual Dynamics scheme, the hierarchy of behaviors is sorted according to time scale: the more long-term and comprehensive a behavior, the higher its position in the hierarchy. Only behaviors at the lowest, "elementary" level have access to actuators.

Each behavior has an activation variable, which is regulated through (many of) the behavior's ODE's. Interactions between behaviors within a single level occur by various types of cross-influences between activations (e.g., mutual inhibition, sequencing). Inter-level interactions are of a quite special sort: higher-level activations work as control parameters on the lower-level subsystems, inducing certain bifurcations in them.

Elementary behaviors are special in that they also generate target trajectories for the robot's actuators. The target trajectories of all elementary behaviors are weighted by the generating behavior's activations, linearly combined and finally passed to the actuators.

This architecture allows to design behavior control systems that are mathematically transparent. The main source of mathematical transparency is the fact that bifurcations of individual subsystems are constrained to certain types (details in [9]). However, mathematical transparency is necessary, but not sufficient for an efficient and scalable practical design. The mathematical constraints which are implicit in the kind of ODE's admitted, must be rendered explicitly manipulable for the concrete design process. This is where the robotics engineer needs a special co-design tool which "embodies" the Dual Dynamics formalism.

3. APICES

The software prototyping environment APICES allows to generate tool skeletons from an object-oriented model of a tool. This tool model is based on a UML [1] class diagram. One of the specific strengths of our environment is its support of graph-based tool structures. We reported on this feature in previous work [3][4]. In this paper, we concentrate on the development process of a co-design tool.

APICES comprises a generic tool template which can be tailored by parameters supplied as annotations of the class diagram. The implementation architecture of the tool template is sketched in the right part of figure 2.

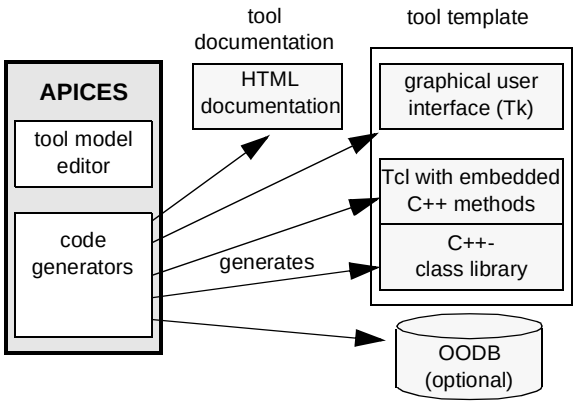

Figure 2. APICES

The code generators of APICES allow to generate a frame of a tool (tool frame) as an instance of a tool template. A tool frame is an application skeleton consisting of a generated C++ class library (implementing the class diagram) embedded into the scripting language Tcl [13]. The embedded C++ library offers methods to construct complex object nets in Tcl.

The tool frame is supplemented with a canvas based GUI which is configured by annotations added to classes of the tool model. The GUI allows to create, modify and edit objects and object relationships via context sensitive menus.

The implementation effort to construct a specialized tool on the basis of a generated tool frame is reduced to adding application specific functionality to the tool frame. We call these extensions tool code in the sequel of this paper. Writing tool code is done either on scripting level by programming in Tcl using generated embedded methods or on C++ level by adding new C++ methods to the class library. Note, that these new methods result in new Tcl commands.

4. DESIGNING DDD WITH APICES

Tool design with APICES is done in three steps in an iterative loop:
(1) capture the object-oriented UML class diagram and the GUI specification in a tool model,
(2) generate the tool frame and the tool GUI from it, and
(3) customize and extend the tool frame with tool code.

4.1 Tool Modeling

In a first design step we captured the notions of Dual Dynamics in a UML class diagram. Since APICES provides a documentation editor as integral part of the model editor, we are able to generate frame-based HTML documentation from the tool model. Figure 3 shows a screen shot of APICES with the DDD tool model.

4.2 Tool Frame Generation

From this tool model we generate a tool frame for the DDD tool. The frame is an instance of the tool template parameterized and tailored by the tool model. The tool frame comes with prefabricated functionality, e.g., for saving and loading object nets, in our case Dual Dynamics models, using a generated file exchange format.

Up to this time, no line of code has to be written manually. All “implementation” was graphical modeling and model parameterization.

4.3 Tool Code

The generated tool frame with the tool GUI is the starting point for adding application specific functionality. We group the tool code required for the DDD tool into two functional parts, an editor and several code generators.

The editor allows to enter a Dual Dynamics specification using graphical elements for sensors, actuators, sensor filters, higher-level behaviors and lower-level behaviors via the generated GUI. Figure 4 gives a screen shot of a very simple Dual Dynamics specification. The editor code to be written comprises assistance functions for automatic incremental layout and check of layered higher-level

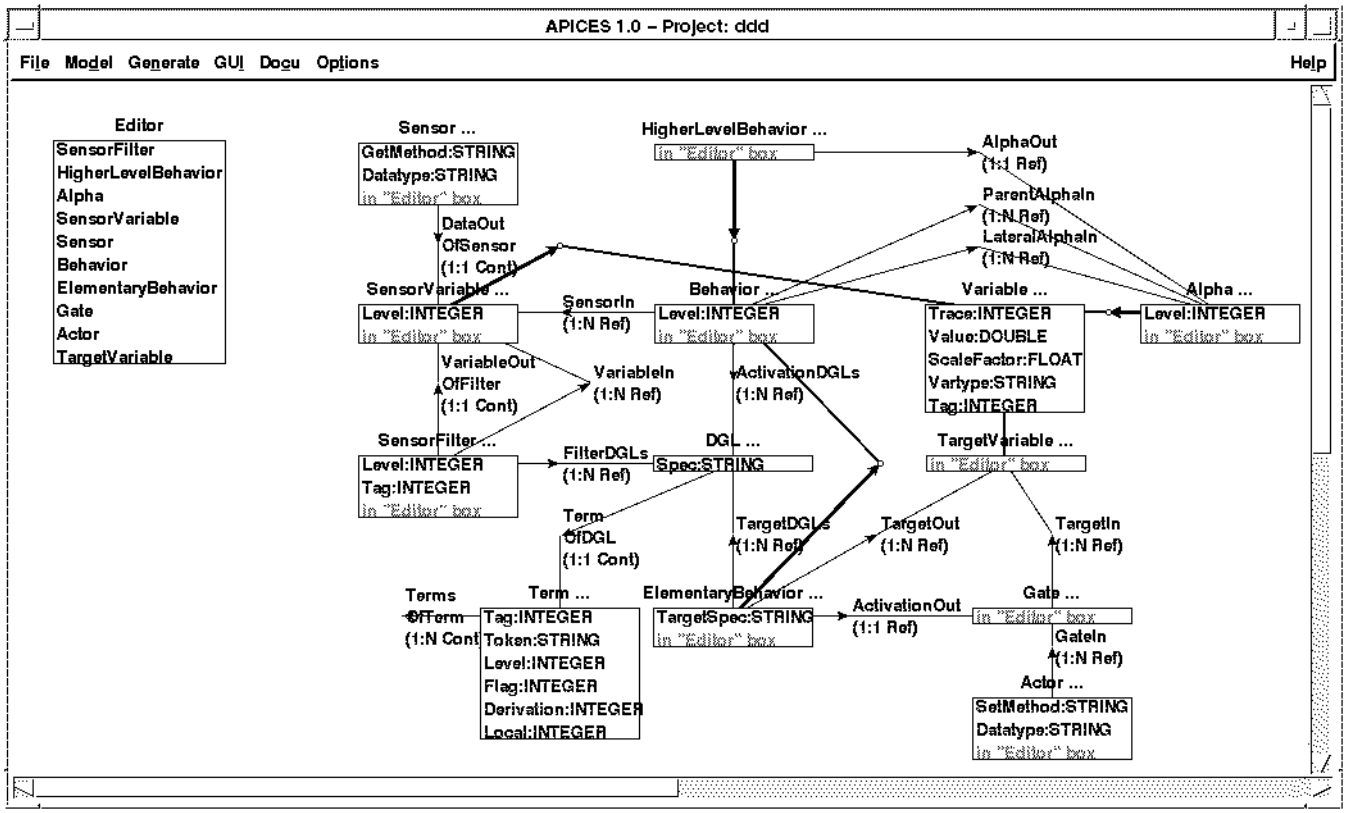

Figure 3. DDD tool model in APICES (left part showing the GUI model)

behaviors and the sensor filter network. The latter allows to highlight variable cycles (figure 4) which give important hints to the behavior designer.

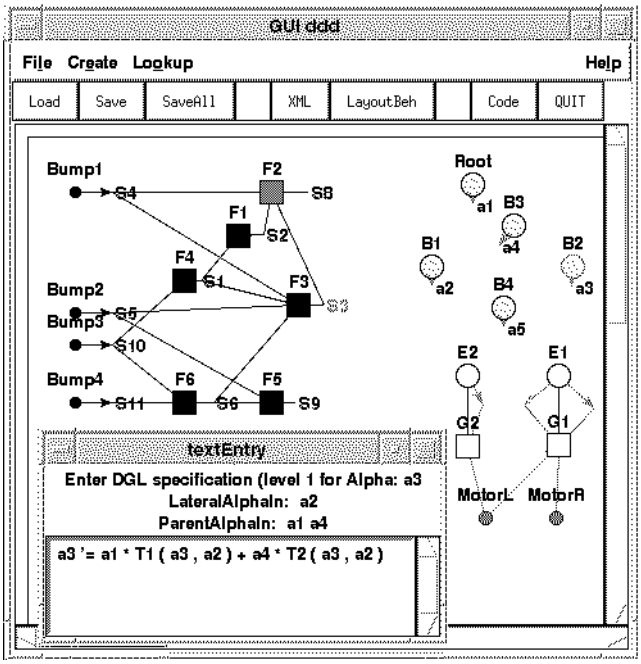

Figure 4. DDD screen shot with ODE editor menu and a variable cycle between sensor variable S3 and sensor filter F2

In addition, we construct a differential equation editor allowing to enter the functional description of sensor filters, higher-level behaviors and elementary behaviors. Differential equations and all other equations are stored as parse tree in the internal data structure of the tool. This allows to perform several syntax checks (well-formedness, variable scopes, parenthesis errors, etc.) on the fly during editing the equations of a Dual Dynamics system.

The code generators convert differential equations to difference equations and produce code for three different targets:
- C code for the three 80C167 micro-controllers used in our robot platform [11] (This code implements the Dual Dynamics system on the robot),
- HTML documentation of the Dual Dynamics system under design, and
- a Java behavior simulation model of the Dual Dynamics system which is plugged into our behavior simulator.

At present, the following extensions are under construction:
- automatic conversion of floating point arithmetic to integer arithmetic for improved micro-controller performance, and
- a model generator for Matlab/Simulink for precise simulation of partial behaviors.

5. RESULTS

We have documented the development process of the DDD tool in order to discover bottlenecks and starting points for future extensions of APICES. Therefore, we can give some figures on the kind of effort spent during the design of DDD (Table 1) and some figures concerning tool code size and implementation time (Table 2).

Kind of design effort	Time effort	Share of effort
knowledge acquisition and tool modeling	15 h	15 %
tool code implementation	45 h	40 %
tool test and bug-fix	20 h	20 %
tool documentation	25 h	25 %

Table 1. Tool design effort

In the first phase of the project, we captured the object-oriented tool model needed for Dual Dynamics. This required a knowledge transfer from the Dual Dynamics expert to the tool designer. The HTML documentation generated from the tool model by APICES turned out to be well-suited to understand and communicate the notions and principles of Dual Dynamics and to explore the relationships between them.

The effort for tool code implementation is detailed in Table 2. It shows the kind of Tcl code written for each sub-system of the tool, either program (P) or configuration data (D). The code generators use a common parse tree traverser which performs three different file export actions for micro-controller C code, for behavior simulation models in Java and for HTML documentation. The total tool code size of DDD currently comprises 1050 lines of Tcl only.

DDD sub-system	Tool functionality	Data Prog.	Lines of Tcl	Time effort
Editor	GUI customization	D	100	2 h
	ODE processor	P	260	10 h
	DD model export (XML)	P	70	5 h
	ODE Editing Assistant	P	80	9 h
	DD File Save/Load	D	30	2 h
Code Generators	C code generator	P	500	17 h
	HTML code generator	P		
	Java model generator	P		

Table 2. Implementation effort

The DDD tool design is recent work in our RoboCup project. Very first user experiences show that the automated Dual Dynamics design process with its convenient behavior editor and the code generators for Java behavior simulation models and the micro-controller C code significantly ease and accelerate robot behavior exploration as well as implementation.

6. RELATED WORK

Commercially available UML-based modeling tools, e.g. Rational Rose [14], or off-the-shelf software development environments, e.g. Visual C/C++, are too general to allow very rapid co-design tool development. For example, they do not offer support for scripting language embedding and automated GUI design. We consider both features combined with a model-based approach essential for rapid tool development.

Nevertheless, modeling tools like Rational Rose are candidates for an alternative frontend of the APICES environment. In this context, we presented work on the integration of APICES with Rational Rose and the source code engineering environment SNiFF+ [5].

7. CONCLUSIONS

In this paper, we show that UML-based tool development with APICES allows to reduce the effort for the specification and construction of highly specialized co-design tools. We have demonstrated this by elaborating on the design of the tool DDD which generates robot control software from a differential equation system.

The DDD tool as described in this paper has been constructed from scratch by generating a tool frame from an object-oriented tool model and extending it with 1050 lines of Tcl. The total effort of the tool design process was three weeks.

The key features of our design process being responsible for this efficiency are:
- rigorous object-oriented top-down modeling,
- re-use of generic tool templates,
- generation of a model-specific tool frame (including GUI) as implementation of the tool model, and
- programming in a scripting language using generated tool model specific language extensions.

8. FUTURE WORK

Our future work on the specialized DDD tool will concentrate on adding an abstraction layer above differential equations which would allow to describe and not construct differential equations. We then will extend the DDD tool in order to assist automatic generation of Dual Dynamics systems from these high-level behavior descriptions.

Our future work on APICES will focus on the abstraction of additional tool sub-systems in order to provide them as re-usable, configurable building blocks in our design tool prototyping environment.

9. ACKNOWLEDGEMENTS

I thank my colleague Herbert Jaeger for offering his expertise in Dual Dynamics.

10. REFERENCES

[1] Alhir, S.S. UML in a Nutshell. O'Reilly, 1998

[2] Balarin, F. et. al. Formal Verification of Embedded Systems Based on CFSM Network. in *Proceedingsof the 33rd ACM/IEEE Design Automation Conference (DAC'96)*, 1996, pp. 568-571

[3] Bredenfeld, A. APICES - Rapid Application Development with Graph Pattern. in *Proceedings of the 9th IEEE International Workshop on Rapid System Prototyping (RSP'98)*, Leuven, Belgium, pp. 25-30, 1998

[4] Bredenfeld, A., and Camposano, R. Tool Integration and Construction Using Generated Graph-Based Design Representations. in *Proceedings of the 32nd ACM/IEEE Design Automation Conference (DAC'95)*, pp. 94-99, 1995

[5] Bredenfeld, A., and Ihler, E. RoseSNiFF - Forward Engineering using Rational Rose and SNiFF+. in *3rd European SNiFF+ Users Conference*, January 1999

[6] Brooks, R.A. Intelligence without reason. *A. I. Memo 1293*, MIT AI Lab, 1991

[7] Buck, T.T., Ha, S., Lee, E.A., and Messerschmitt, D.G. Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems. in Int. *Journal of Computer Simulation*, special issue on "Simulation Software Development", vol. 4, pp. 155-182, April, 1994.

[8] Chou, P., Ortega, R.B., and Borriello, G. The Chinook Hardware/Software Co-Synthesis System. in *Proceedings of the Eight International Symposium on System Synthesis*, 1995, pp. 22-27

[9] Jaeger, H. and Christaller, T. Dual Dynamics: Designing Behavior Systems for Autonomous Robots. in *Artificial Life and Robotics* (to appear)

[10] Kitano, H. Research Program of RoboCup, *Applied Artificial Intelligence.* Vol. 2, No. 2-3, pp. 117-126, May 1998

[11] Kuth A., et al. Team Description of the GMD RoboCup-Team. in *Proceedings of the 2nd RoboCup Workshop*. Edited by M. Asada, Paris, pp. 439-450, July 1998

[12] Madsen, J., et. al. LYCOS: the Lyngby Co-Synthesis System. *Design Automation of Embedded Systems.* Vol. 2, No. 2, March 1997

[13] Ousterhout, J.K. Tcl and the Tk Toolkit. *Addison-Wesley*, Reading, MA, 1994

[14] Rational Rose98/C++. Rational Software Corp., Santa Clara, CA, 1998

[15] Valderrama, C. A., et. al. A Unified Model for Co-Simulation and Co-Synthesis of Mixed Hardware / Software Systems. in *Proceedings of the European Design and Test Conference*, 1995, pp. 180-184