SEAMLESS INTEGRATION OF RECONFIGURABLE HARDWARE INTO THE ROBOTIC DEVELOPMENT PROCESS

Moritz Schilling

DFKI GmbH, Robert-Hooke-Str. 5, 28359 Bremen, Germany

TEC-MMA, ESTEC, 2200 AG Noordwijk, The Netherlands

ABSTRACT

Modern robotic development frameworks have agreed upon the model-based, component-centric software development approach. Composing robotic control software has become an easier and less error-prone task. While there is a model for the software architecture and fairly recently a model for the kinematic structure of robotic systems, models for other aspects are missing. This paper will focus on the model of heterogeneous computational resources and their interconnection within a robot and outline how tools make use of the models to generate code skeletons in different languages, find compatible implementations of algorithms given a specific target and to map a network of software to a network of hardware components.

1. INTRODUCTION

Robotic systems are becoming more and more complex either in mechanical, electronic or software domain. Regarding electronic and software domain this complexity mainly arises from the composition of different heterogeneous processing devices and their interconnections. How this heterogeneity in hardware can be transformed to homogeneity in the software domain has gained much attention in research - but mostly outside of the robotics community. As robotic systems are also distributed heterogeneous embedded systems the development of robot control systems will benefit from these research efforts. For example, the development process itself will be accelerated, the usage of resources will be more efficient and/or the robustness of the system against failure will increase.

While there has been much progress in unifying the programmability of networks of heterogeneous, conventional computing devices, reconfigurable hardware devices (e.g. Field Programmable Gate Arrays) still have to be treated separately. How such devices can be seamlessly incorporated into robotic development frameworks which have been developed to improve the re-usability of software and to ease the programming of robotic systems is the main focus of this research. Such a framework shall be enabled to make use of the advantages of reconfigurable hardware, while maintaining system stability, integrity and predictability.

In the following a review of the state of the art with respect to the programming of heterogeneous processing devices in embedded systems like robots is presented. Besides robotic development frameworks a look into other related areas of research is taken as well.

The considered frameworks have been selected either because of their popularity amongst the robotics community or because they exhibit an outstanding feature:

ROCK the Robot COnstruction Kit which is mostly used by DFKI and ESTEC and runs on robots like SpaceClimber, SpaceBot or Asguard[ROC15],

BG the Behaviour Graphs which have been recently developed at DFKI to model the reactive layer of robots like SpaceClimber or Charlie[Lan12],

ADE a framework from the University of Notre Dame which has been used to control the famous Nao robots[Sch],

GenoM3 the Generator of Modules tool-set developed at LAAS-CNRS and is used to design real-time software architectures in the robotic and space domain[LC15],

ROS the Robot Operating System which is one of the most popular frameworks in the robotics community[ROS15], and

LabView which is not a pure robot development framework but has been used for this purpose[Mue11].

These frameworks are analysed with respect to their modelling capabilities in the relevant domains of software and computational hardware, the portability of the generated deployments which affects heterogeneity and the levels of transparency which affect the possibility to distribute a given application throughout the system.

Modelling Capabilities The frameworks are analysed with respect to their modelling capabilities of *sensors*, *actuators*, *computational resources* and *network topology*. These capabilities are essential for distributed systems support. Other important modelling capabilities like mechanical structure, kinematics etc. are not considered here.

Portability Because this project is about distributed, *heterogeneous* systems, the frameworks have to be analysed with respect to their support of different processing devices. Hereby, it is of special interest if the frameworks not only support conventional processing devices - like *CPU*s or μ*C*s - but also support reconfigurable hardware, namely *FPGA*s.

Transparency Transparency specifies which implementation details are hidden from the user such that he/she doesn't need to care about.

Communication transparency hides the details of data transport from the user,

Transaction transparency hides the coordination of dependent components from the user,

Location transparency hides the location of a component in the system from the user,

Migration transparency hides the relocation of components in execution from the user,

Replication transparency hides the cloning of components from the user,

Ressource transparency hides the resource management from the user,

Persistence transparency ensures that the creation/destruction of components do not affect independent components, whereas

Failure transparency hides the failure and recovery of components during execution.

A detailed and formal description of these transparency concepts can be found at [RM-15].

Transaction transparency is investigated in more detail with respect to the execution semantics which specifies under what conditions a component is to be executed. At the one extreme there are execution models which make the constructable applications *decidable* and *deterministic* while on the other end there are those which make them *undecidable* and *non-deterministic*. To get a categorization, three well-known models have been chosen:

SDF Static Data Flow (decidable and deterministic)

KPN Kahn Process Networks (undecidable and deterministic)

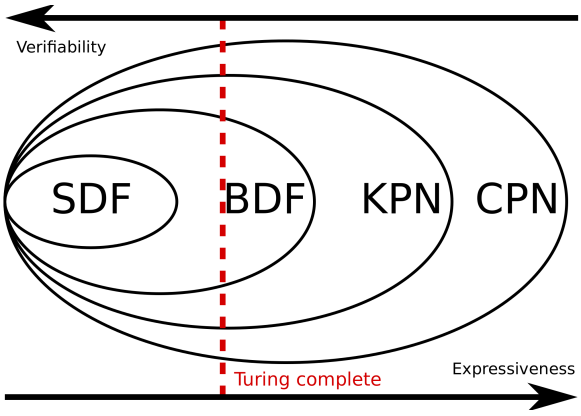


Figure 1. The relationship between verifiability and expressiveness for a selection of computational models. BDF stands for Boolean Dataflow and extends the SDF domain with data-dependent routing capabilities. Based on [Bas].

CPN Coloured Petri Nets (undecidable and non-deterministic)

The execution model directly influences the possibility of verification and the expressiveness of the component networks. The more expressive the less verifiable such a network becomes. Figure 1 visualizes this relationship.

Evaluation ROCK components comprise an interface consisting of inputs, outputs and parameters (properties) with data types derived from C++ types. Furthermore, components have a life-cycle which is realized by a finite-state-machine whose state is published to the environment and can be controlled by the user. Therefore there exist mechanisms to automatically recover from failure by monitoring the state of components. Semantics roughly follow the KPN model; components are connected peer-to-peer through channels and are accessed by read/write functions. However, the user can (but shouldn't) circumvent the KPN model because emptiness of a channel can be tested. ROCK uses message queues or CORBA for data transport dependent on the location of the interacting components. Components can be deployed to systems with conventional processing devices capable of hosting a UNIX OS.

Behaviour Graph components (nodes) exhibit inputs and outputs with C float type. The granularity of the atomic components is fine-grained; they are pure mathematical operators (like add, multiply etc.). As such, their semantics follow the SDF model. Only if all of their operands are available operations are executed. Currently, behaviour graphs can only be executed on CPUs or μCs and only the coordination of the components is hidden from the user.

ADE components have four types of interface links,

- activation links,

- observation links,
- process control links, and
- component links.

Activation links are used to exchange data between components, observation links pass state information to other components while process control links can change state, and component links allow components to create new components. Because of these dynamic properties, e.g. the creation and interconnection of components during run-time, the semantics of a component follow the CPN model in which non-determinism is possible. Communication relies on registries, servers and clients and is implemented in JAVA language as well as the components itself. Therefore, an ADE component needs a JAVA virtual machine on top of conventional processing device (and possibly an OS). This virtualisation and the component link mechanism provide replication transparency. ADE also models computational structure of the system in form of virtual machines connected by communication links.

A GenoM3 component consists of inputs and outputs of data types defined in the Interface Description Language of CORBA. Additionally, it defines events which are used e.g. to change state of the component. The component behaviour is directly modeled as a Petri Net; so the semantics is of the CPN domain. GenoM3 uses middleware templates to specify the glue code necessary for communication, distribution etc. Therefore, all other properties strongly depend on these templates. For example, a VHDL template could be developed which would enable GenoM to support reconfigurable hardware. However, to the knowledge of the author such templates do not yet exist.

ROS components do not have the notion of inputs and outputs but use the concept of topics. Topics are anonymous channels to which a ROS component can subscribe to or to which it can publish information. As such, components are not directly coupled as in channel-based component networks. Instead, they form a component network at run-time. This process enables non-deterministic behavior because it is not known which component is connected to which of the others beforehand. The semantics are therefore of the CPN domain. As most of the previously presented frameworks, ROS also provides glue code for communication which is based on the TCP/IP stack. The components are deployed and executed only on CPUs hosting an OS.

Although LabView is not a robotic development framework of its own, it has been used for this purpose. The components of LabView follow the SDF semantics like those of the Behaviour Graph framework. But the available components allow much more than purely mathematical operators; there are also components which exhibit data-dependent behaviour like loops and if-then-else structures. LabView also provides the necessary glue code to let the components interact and it is the only framework considered here which is able to deploy them to FPGAs. However it remains unclear if distributed operation in heterogeneous processing networks is supported by the models and the tool-chain.

Table 1 summarizes the properties of the different frameworks. Most frameworks exhibit a very powerful and expressive component model while only ROCK, Behaviour Graphs and LabView use a more restrictive one. As has been stated, more expressiveness comes at the price of less verifiability. However, the necessary tools for verification are still missing in all of the frameworks. When it comes to critical applications like space robotics this aspect can not be neglected.

Only one framework supports different kinds of processing devices and (therefore) different programming languages. Modern robots consist of various processing devices hosting either a CPU, a microcontroller, an FPGA or combination of them. It is therefore essential to allow the programming of these devices and the data exchange within the framework.

To be able to distribute a given application to more than one target in a processing network the topology of it has to be modeled as well. To the knowledge of the author only ADE considers such a model for distributing applications. Furthermore, ADE allows dynamical redistribution during run-time which is a prerequisite to guarantee robustness against failures and load balancing.

None of the frameworks exhibits all of the properties necessary for the programming of heterogeneous and distributed robotic systems which are

- distributed computation
- verifiability
- heterogeneous processing networks
- dynamic reprogramming (not reconfiguration via parameter updates)

2. DISTRIBUTED BEHAVIOR GRAPHS

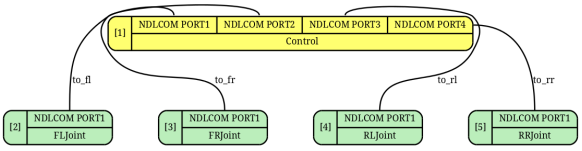
The behavior graph formalism has been chosen as a first candidate to show the feasibility of wholistic programming of heterogeneous robots. Because of their simplicity, the basic building blocks or atomic components are easily transferable to the FPGA-domain. The atomic components of the behavior graphs are

- N-ary reduction operations like $\Sigma, \Pi, ||.||_2$
- Trigonometric operations ranging from $sin(x)$ to $atan2(y, x)$
- Other transcendental operations like $log(x)$ and x^y
- Ternary if operation (*C-syntax: x == c ? a : b*)

Framework	Models	Portability	Transparency
ROCK	Sensors, Actuators	CPU & OS	Communication, Transaction, Location, Resource Persistence, Failure
Behavior Graphs		CPU,μC	Transaction
ADE	Sensors, Actuators Resources, Topology	CPU & JVM	Communication, Transaction, Location, Resource Persistence, Failure, Replication
GenoM3		Depends	Depends
ROS	Sensors, Actuators	CPU & OS	Communication, Transaction, Location, Resource
LabView	Sensors, Actuators	CPU, FPGA	Communication, Transaction

Table 1. Evaluation of the different frameworks.

(a) ... as a CAD model



(b) ... and a possible model of its computational infrastructure

Figure 2. A rover ...

Through composition, more complex mathematical operators can be formed from the basic operations. However, the formalism is not Turing-complete, because *WHILE* loops can not be constructed given the synchronous dataflow like semantics.

To model the robot hardware, a library called *hwgraph* for constructing and representing computational hardware as graphs has been developed. These hardware graphs currently support hierarchy, heterogeneity of processing elements and interfaces and point-to-point connections. It has been designed such that other topologies, processing elements and interfaces can be added to the model easily. Figure 2 shows a rover as a CAD model and a possible computational infrastructure modelled with this library. There is a central node *Control* of type *CPU* and four joints of type *FPGA*, each connected to the central node via a point-to-point connection of *NDLCom* interfaces (for details refer to [MZS16]).

Let the left part of figure 3 be a behavior graph modelling the desired control of the presented rover. In order to exploit all computational resources and to keep latencies of control loops as short as possible, the constituents of the control have to be assigned to the processing units such that

- inputs and outputs in software match data sources and sinks in the hardware model,
- computational resources are shared suitably amongst the parts of the software, and
- communication between processing units is kept at a minimum.

These constraints are modeled by cost functions which

1. have to be positive semi-definite,
2. are target dependent in general, and
3. must not be dependent on the order in which operations have been assigned to targets.

Figure 3 shows how a partition/sub-graph (a group of software parts) is formed and mapped to an execution target. In this example only target-independent cost functions have been used. One cost function increases with the number of entering or leaving edges to/from a sub-graph, one increases with the total number of software nodes a sub-graph contains. Partitioning and mapping is performed in two steps: the first is to find a good initial mapping, the second is to optimize this mapping if desired. The greedy k-way graph partitioning algorithm presented in [JSB00] has been used to find such an initial mapping. Afterwards optimization algorithms can be applied which check if either the re-assignment of single nodes or the permutation of whole partitions lead to global cost reduction.

Given a behavior graph and a hardware graph the mentioned algorithm produces a sub-graph for each execution unit - these contain all the nodes which are about to be executed on the same device. The resulting mapping and the different types of the execution units are then used by a code generator to either produce *C* code for conventional targets or *VHDL* code for FPGAs. The generated code as well as routing tables for the communication layer are stored in a dictionary. Source code templates for different devices are then filled with the information from the dictionary. These templates cover the instantiation of the behavior graph itself (most inner layer), the separation of internal and external data sources, the de-serialization of incoming data packets and the serialization of outgoing packets, the triggering of the execution/evaluation of the

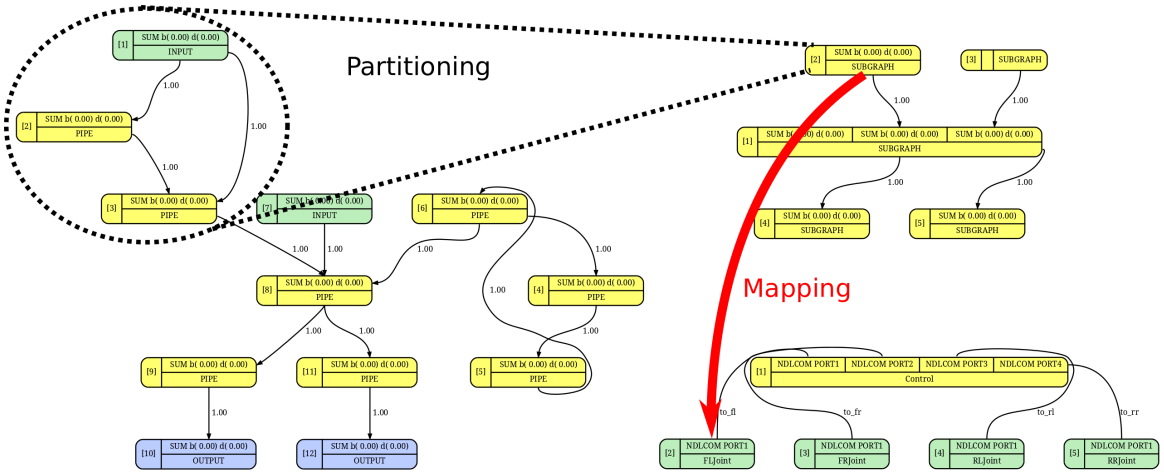


Figure 3. The behavior graph on the left is partitioned (Nodes 1,2,3 *to Node* 2*) to a graph of sub-graphs in the upper right. One of these sub-graphs has been formed for and mapped to an FPGA (Node* FLJoint *in the lower right). Partitioning and mapping are dependent because the cost functions are target dependent in general.*

graph (middle layer), the setup of all external interfaces according to the hardware model, the initialisation of the routing layer, and the reception/transmission of behavior graph data packets.

3. PRELIMINARY RESULTS

The first experiments have been performed using a conventional x86-based personal computer (PC), a Xilinx Spartan6 based and a ARM7 based printed circuit board (PCB) which are interconnected in a chain. Figure 5 shows photographs of the PCBs and the device interconnection. The edges denote communication links for data exchange via the NDLCom protocol which has been developed by DFKI and is used in various robotic systems.

The software which has been used for testing is the behavior graph shown on the left of figure 4. It resembles an audio processing network which implements the famous phaser effect used by C3PO in the Star Wars movies. It mainly consists of an eight stage allpass filter network with feedback. The inner workings of the allpass filters are shown on the right hand side of the figure; the audio input is split into a lowpass (shown in the upper right) filtered and a non-filtered stream which are then fed into an (ideal) operational amplifier (shown in the lower right).

To find out about the transfer function of the network, a signal with a uniform distribution of frequencies or white noise respectively is streamed into it. The filtered signal should exhibit three sharp peaks and four notches in the spectrum. Figure 6 is showing, that this can be observed in the frequency range below 5 kHz. The left figure represents the spectrum of the undistorted white noise, while the right shows the filtered signal. The peaks have been highlighted by red lines.

4. CONCLUSIONS & OUTLOOK

The distributed behavior graph framework is able to generate and distribute networks of simple, fixed mathematical operators to a network of distributed processing units. But to be useful for general purpose application aside from signal processing the instruction set itself should be adaptable either to store already generated sub-graphs, to allow more powerful, possibly non-deterministic operators or multiple implementations for the same algorithm/instruction.

The behavior graph model and the hardware model are based on graphs but there is no underlying model (yet) which allows both modelling domains to be easily incorporated into higher-level domains (such as the mapping domain). A fundamental model would also enable the possibility of extending the different views on a robotic system by e.g. adding models of the electrical wiring or the kinematic structure.

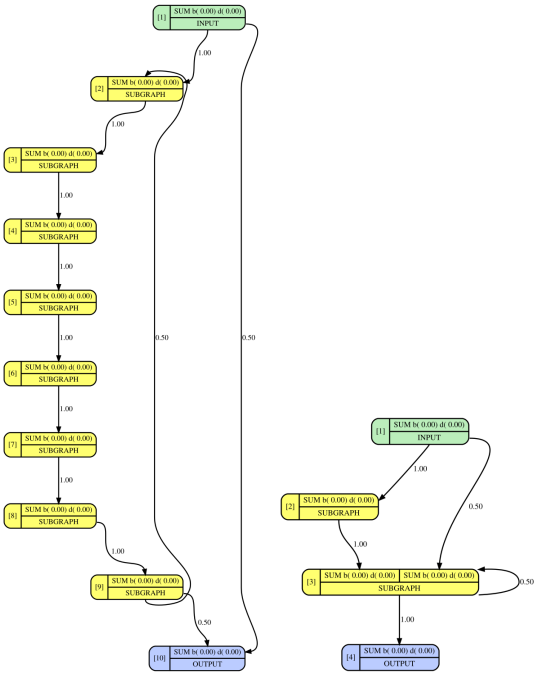
It is planned to utilize ontologies to serve as the basis for the different domains. This will also enable the framework to ensure certain constraints between e.g. the interconnection of devices or software components.

REFERENCES

[Bas] Twan Basten. Kahn process networks and a reactive extension. Summer School on Models for Embedded Signal Processing Systems.

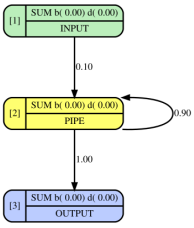
[JSB00] Sachin Jain, Chaitanya Swamy, and K. Balaji. Greedy algorithms for k-way graph partitioning, 2000.

[Lan12] Malte Langosz. A behavior-based library for locomotion control of kinematically complex robots. In *Proceedings of the 16th International Conference on Climbing and Walking*

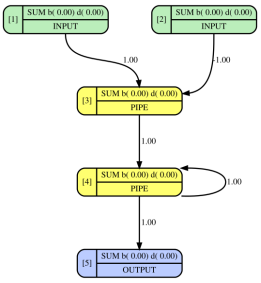


(a) Behavior graph of a phaser.

(b) The subgraphs are allpass filters.



(c) The allpass filter uses a lowpass ...



(d) ... and an ideal opamp.

Figure 4. Overview of the behavior graph used for testing.

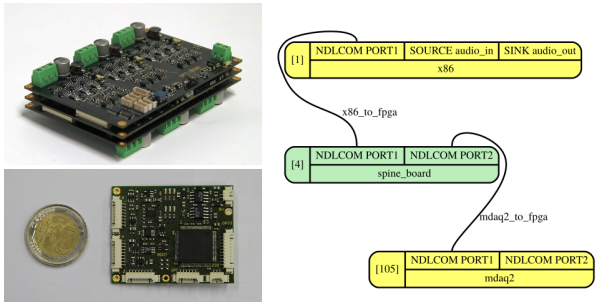


Figure 5. The hardware setup for the preliminary tests consisting of a standard x86 PC (not shown explicitely), an Xilinx Spartan6 based and an ARM7 based PCB connected in series.

Robots and the Support Technologies for Mobile Machines, 2012.

[LC15] LAAS-CNRS. Genom3, 2015.

[Mue11] Karl Muecke. *A Distributed, Heterogeneous, Target-Optimized Operating System for a Multi-robot Search and Rescue Application*. Springer Verlag Berlin Heidelberg, 2011.

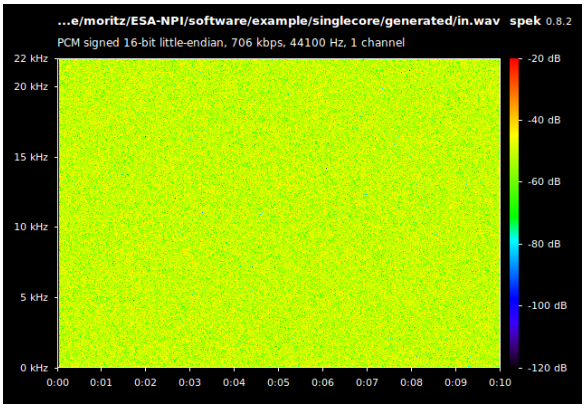
[MZS16] Tobias Stark Martin Zenses, Peter Kampmann and Moritz Schilling. Ndlcom: Simple protocol for heterogeneous embedded communication networks. In *Proceedings of the Embedded World Exhibition & Conference*, 2016.

[RM-15] Reference model of open distributed processing, 2015.

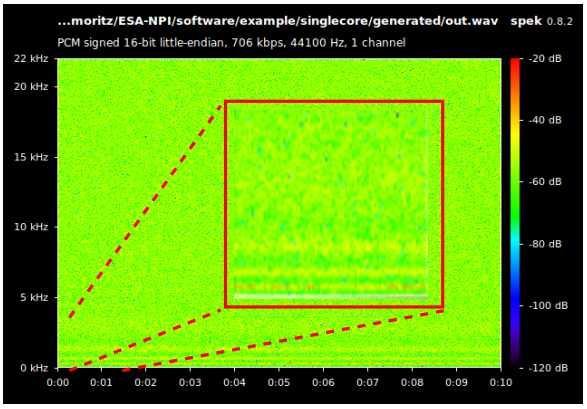
[ROC15] Robot construction kit, 2015.

[ROS15] Robot operating system, 2015.

[Sch] Matthias Scheutz. Ade - steps towards a distributed development and runtime environment for complex robotic agent architectures. University of Notre Dame.



(a) Spectrum of original white noise test signal



(b) Spectrum of filtered signal

Figure 6. Spectra of original and filtered signal over time. In the filtered signal the peaks of the transfer function can be observed for signals below 5 kHz.