

ABSTRACT

Title of thesis: A NETWORKED DATAFLOW SIMULATION ENVIRONMENT FOR SIGNAL PROCESSING AND DATA MINING APPLICATIONS

Stephen Won, Master of Science, 2012

Thesis directed by: Professor Shuvra Bhattacharyya
Department of Electrical and Computer Engineering
University of Maryland at College Park

In networked signal processing systems, dataflow graphs can be used to describe the processing on individual network nodes. However, to analyze the correctness and performance of these systems, designers must understand the interactions across these individual “node-level” dataflow graphs — as they communicate across the network — in addition to the characteristics of the individual graphs.

In this thesis, we present a novel simulation environment, called the *NS-2 – TDIF SIMulation* environment (*NT-SIM*). NT-SIM provides integrated co-simulation of networked systems and combines the network analysis capabilities provided by the Network Simulator (*ns*) with the scheduling capabilities of a dataflow-based framework, thereby providing novel features for more comprehensive simulation of networked signal processing systems.

Through a novel integration of advanced tools for network and dataflow graph simulation, our NT-SIM environment allows comprehensive simulation and analysis of networked systems. We present two case studies that concretely demonstrate the

utility of NT-SIM in the contexts of a heterogeneous signal processing and data mining system design.

A NETWORKED DATAFLOW SIMULATION
ENVIRONMENT FOR SIGNAL PROCESSING
AND DATA MINING APPLICATIONS

by

Stephen Won

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master of Science
2012

Advisory Committee:
Professor Shuvra Bhattacharyya, Chair/Advisor
Professor Silvina Matysiak
Professor Uzi Vishkin

© Copyright by
Stephen Won
2012

Acknowledgments

Without the people in my life, this thesis would not be possible. I would like to give my thanks here.

First and foremost, I would like to thank my advisor, Dr. Shuvra Bhattacharyya, for his enthusiasm, advice, and financial support. From his programming class to my years in graduate school, he has been instrumental in my professional development through internships and research opportunities. Words cannot fully express the appreciation I have for my advisor.

For my thesis defense, it is my pleasure to thank my advisory committee members, Dr. Uzi Vishkin and Dr. Silvina Matysiak, for their time in serving on my thesis committee, but also my experiences in their courses. Both of their courses were two of the most interesting and rigorous courses that I could have experienced in my student career.

For my graduate research, I would like to thank the members of the DSPCAD group for their support and relaxed environment. Special thanks go out to Dr. Chung-Ching Shen, who was instrumental in guiding the research topic of this thesis. I would also like to thank Kishan Sudusinghe for his help in the machine learning research. I am grateful to both Dr. Nimish Sane and Dr. William Plishker, who assisted my undergraduate research. I am also grateful to other members of the DSPCAD group, including George Zaki, Scott Kim, Lai-Huei Wang, Zheng Zhou, Hsiang-Huang Wu, Shenpei Wu, and Inkeun Cho.

For my professional development, I would like to thank all my co-workers

at the Air Force Research Laboratory, Army Research Laboratory, and Patuxent River Naval Air Station. Each site gave me an experience that helped me develop my programming and leadership skills, which were instrumental in my research. I would like to thank Dr. Gunasekaran Seetharaman for giving me an opportunity to experience AFRL in Rome, NY and for supporting me here in ARL and UMD. I would also like to thank Dr. Susan Young for giving me guidance in research. To all my supervisors and coworkers at Pax River, the programming experience and the stories I listened to prepared for college in a way that no other place or person could.

For my personal sanity, I would like to thank all of my friends for the support and fun that they have provided me during my studies. Special thanks goes to Leonardo Apolonio, who gave me proper motivation to delve into the world of machine learning. I would also like to thank Matt Lentz, for being my graduate study and work partner. To Mark Finkelstein, thank you for your hijinks and showing me how important it is to constantly push the envelope. I would like to thank Elizabeth Kenyon for answering my questions about work. To the people that I have lived with, thank you for constantly trying to get me to experience new things.

Lastly, I wish to thank my parents, Chi and Susan Won. No words can fully express the appreciation and gratitude that I have for them. From raising me with a proper technical background to supporting some of my crazy adventures, I dedicate this thesis to them.

The research presented in this thesis has been sponsored in part by the US Air Force Office of Scientific Research (AFOSR), and US Air Force Research Laboratory (AFRL). Any opinions, findings, conclusions, or recommendations expressed in this

thesis are those of the author and do not reflect the views of AFOSR or AFRL.

Table of Contents

List of Figures	vii
List of Abbreviations	ix
1 Introduction	1
1.1 Contributions of this thesis	2
1.2 Outline of the thesis	4
2 Background	6
2.1 Dataflow Modeling	6
2.2 Dataflow Interchange Format (DIF)	7
2.2.1 The DIF Package (TDP)	7
2.3 Functional DIF	8
2.3.1 Targeted DIF (TDIF)	9
2.3.2 Lightweight Dataflow Environment (LIDE)	11
2.4 Network Simulator	12
3 Related Work	14
4 Networked Dataflow Application Simulation	18
4.1 NT-SIM Application Design Framework	18
4.1.1 NL-SIM Application Design Framework	20
5 Case Study: Image Registration Sensor Network	22
5.1 Distributed Vision Sensor Systems	23

5.2	Actor Design	25
5.3	Actor Separation at the Node Level	26
5.4	Network Creation	28
5.5	Simulation of the Distributed System	30
6	Case Study: Adaptive Stream Mining	36
6.1	Support Vector Machines	36
6.2	Actor-Level Design	38
6.3	Subsystem Level Design	40
6.4	Network Design	41
6.5	Simulation of the Adaptive Stream Mining System	42
7	Conclusions and Future Work	44
	Bibliography	46

List of Figures

2.1	TDP-based design flow.	8
2.2	TDIF-based design flow.	12
4.1	Specifying the interaction between dataflow applications and network simulations in NT-SIM.	19
4.2	A simple example of an NT-SIM networked application system model. In this model, two different nodes perform the addition operation. IAs passing information from and to the Network Object block in the ns-2 subsystem are shown as yellow actors.	20
5.1	A dataflow graph model of SIFT-based feature detection and image registration.	24
5.2	Dataflow graph subsystem for SIFT feature detection.	27
5.3	Subsystem for image registration after SIFT feature detection.	27
5.4	The topology represented by the Tcl script for the SIFT sensor network.	31
5.5	Graphical representation of the simulated SIFT sensor network.	32
5.6	Reference image used for SIFT feature detection.	33
5.7	Target image used for SIFT feature detection.	34
5.8	Resulting registered image from SIFT VSN case study using NT-SIM.	35
6.1	A simple example of classification using a linear support vector machine. Points on the boundary represent the support vectors.	37
6.2	Dataflow graph subsystem for reading in face images.	41

6.3	Subsystem for SVM classification.	41
6.4	Graphical representation of the simulated face detection network. . .	43

List of Abbreviations

ACM	Adaptative Classifier Manager
API	Application Programming Interface
ASM	Adaptive Stream Mining
BDF	Boolean Dataflow
CFDF	Core Functional Dataflow
CSDF	Cyclo-Static Synchronous Dataflow
DICE	DSPCAD Integrated Command Line Environment
DIF	Dataflow Interchange Format
EC	Execution Context
FIFO	First-In-First-Out
FPGA	Field-Programmable Gate Array
GPU	Graphics Processing Unit
HSDF	Homogeneous Synchronous Dataflow
IA	Interface Actor
LIDE	Lightweight Dataflow Environment
NL-SIM	NS-2-LIDE Simulation
NT-SIM	NS-2-TDIF Simulation
OSI	Open Systems Interconnection
RANSAC	Random Sample Consensus
SDF	Synchronous Dataflow
SVM	Support Vector Machine
TC	Topological Context
TCP	Transmission Control Protocol
TDIF	Targeted Dataflow Interchange Format
TDP	The DIF Package
UDP	User Datagram Protocol

Chapter 1

Introduction

Multimedia and data mining applications often require intensive stream processing capabilities to maintain performance constraints. To increase the performance and capabilities of the applications, diverse platforms and devices are employed, including programmable digital signal processors, microcontrollers, graphics processing units (GPUs), and field programmable gate arrays (FPGAs). This creates heterogeneous computing environments that can utilize networks of different computing devices. However, this leads to different programming models and development environments, which makes programming to these environments a challenging task. To help address this challenge, dataflow models of computation can be used to describe the signal processing or data mining functionality due to their formal correspondence with signal flow graphs and exposure of high level computational structure within the application.

In dataflow models of computation, applications are represented by directed graphs known as dataflow graphs. In a dataflow graph, vertices (*actors*) represent computational modules for running (*firing*) computational tasks and edges represent first-in-first-out (FIFO) channels that store data values (*tokens*) and establish data dependencies between actors. The actors can produce or consume tokens from their respective output and input edges, with the firing controlled by the availability of

data in the input edges.

This framework allows scheduling to benefit from the dataflow graph of the application. The process of determining the firing conditions and sequencing the actors to share limited processing resources is called *scheduling*. In addition to establishing data dependencies, scheduling can be used to exploit parallelism to improve performance and utilize memory efficiently for buffer management. This leads to a variety of techniques for dataflow graphs to achieve different objectives, including latency optimization, throughput optimization, buffer management efficiency, and adaptive scheduling flexibility (e.g., see [1, 2, 3, 4, 5, 6, 7]).

However, dataflow models are not typically applied to the networking aspects of networked applications. Simulations involving data protocols and link conditions are usually not represented by dataflow techniques, as such functionality involves significant control and discrete event behavior. Co-simulators can simulate both network conditions and application settings at each node. However, conventional cosimulation methods do not utilize dataflow graphs for intra-node modeling of the application. Combining this capability helps to provide complete system analysis of networked signal processing or data mining applications without sacrificing the benefits of dataflow-based design frameworks at the level of individual nodes.

1.1 Contributions of this thesis

In this thesis, we present a hierarchical design method to model network conditions and dataflow models of signal processing and data mining applications. To

do this, we developed the *NS-2 – TDIF SIMulation (NT-SIM)* and *NS-2 – LIDE SIMulation (NL-SIM)* co-simulator environments.

The NT-SIM and NL-SIM co-simulators are platform-independent tools that integrate their respective dataflow modeling environments with Network Simulator (*ns-2*) [8] to ensure deterministic actions in networked signal processing and data mining systems. Both NT-SIM and NL-SIM allow the designer to focus on the dataflow model at the application node level while *ns-2* allows analysis of network properties and information sharing among nodes in a distributed application. This allows the designer to apply powerful dataflow-based optimization techniques for signal processing systems at the network node level (e.g., see [7]), and to accurately simulate network interactions across the optimized node-level implementations using the state-of-the-art network simulation capabilities of *ns-2*.

As part of this thesis, we present two case studies. We first demonstrate our framework on a sensor network for image registration. Image registration benefits from a heterogeneous environment, where more compute-intensive operations can take advantage of specialized hardware such as GPUs. Our simulation framework helps to simulate accurate interactions between optimized node level software that exploits the GPUs, and the network-level interactions that influence the environment in which the nodes operate.

The second case study deals with an exploration of multiple classifiers in a data mining application, where we are trying to detect faces from sets of input images. In such an application, the designer may choose to deploy multiple classifiers with different points of operation, and have the classification method selected dynamically

based on input data characteristics or operational constraints. It is important to establish deterministic behavior across different environmental conditions to ensure the optimal performance using multiple classifiers, and the restrictions imposed by signal processing oriented dataflow models of computation help to enhance such deterministic, real-time operation. With this demonstration of NL-SIM, we show how the selection of multiple classifiers on different nodes can be performed to optimize for different conditions. Furthermore, we show how such design considerations can be simulated in the context of their overall network-level embedding, not just in isolation.

Through their novel integration of capabilities for dataflow-based, intra-node design, and accurate, efficient network-level simulation, NT-SIM and NL-SIM allow designers of networked signal processing systems to systematically optimize implementations, and validate deterministic operation in a distributed setting.

1.2 Outline of the thesis

The rest of the thesis is organized as follows. Chapter 2 discusses background and related work on dataflow graph modeling and network simulation. Chapter 3 introduces our co-simulation approach, and provides a formalization of the heterogeneous design approach for our proposed networked dataflow simulation environment. Chapter 4 describes a case study in which we use NT-SIM for a sensor network that is designed to perform image registration. Chapter 5 presents a second case study, which involves networked data mining of heterogeneous machine learning classifiers

using NL-SIM. Finally, conclusions and future work are discussed in Chapter 6.

Chapter 2

Background

In this chapter, we cover background on modeling applications using dataflow graphs, and on network simulation tools.

2.1 Dataflow Modeling

Coarse-grained dataflow graphs are widely used in the digital signal processing (DSP) community to model DSP applications. As a result, a wide variety of dataflow models and tools have been developed to suit various application needs (e.g., see [7]). Dataflow modeling techniques provide rich trade-offs among expressive power, analysis potential, suitability to different types of signal flow graph structures, and available optimization techniques [7]. Developers of dataflow based signal processing applications match their applications with available dataflow models, typically starting with consideration of basic models, such as synchronous dataflow (SDF), homogeneous synchronous dataflow (HSDF), and cyclo-static dataflow (CSDF) [9], and moving if needed to more expressive models, such as Boolean dataflow (BDF) [10] or enable-invoke dataflow [11].

For such an application model, we define a dataflow graph G , which is comprised of an ordered pair (V, E) , where V is the set of actors representing computations and E is the set of directed edges representing FIFO communication links

between nodes that store tokens. A directed edge $e = (v_1, v_2) \in E$ is comprised of a source node $v_1 \in V$ and a sink node $v_2 \in V$. Each actor can consume and produce tokens from its respective input and output edges in a single firing, which is controlled by the availability of tokens in the input edges.

2.2 Dataflow Interchange Format (DIF)

The dataflow description of the application can be described using the Dataflow Interchange Format (DIF), a language used to describe the dataflow semantics of an application [12]. DIF allows the designer to describe dataflow-related and actor-specific information about the application at different levels of granularity and hierarchy. Since dataflow semantics are platform and design tool independent, the dataflow semantics used to describe an application can be specified uniformly in a heterogeneous compute environment using DIF. Moreover, DIF provides syntax to allow for platform- or tool-specific information to be captured in intermediate representations of the application.

2.2.1 The DIF Package (TDP)

To use the semantics captured by a DIF description of an application, the DIF package (*TDP*) was created. An overview of TDP is shown in Figure 2.1. TDP transforms a DIF description into an internal representation where TDP's graph utilities, optimization engines, and other algorithms can exploit the dataflow properties of the application. This makes TDP a suitable environment to model dataflow applica-

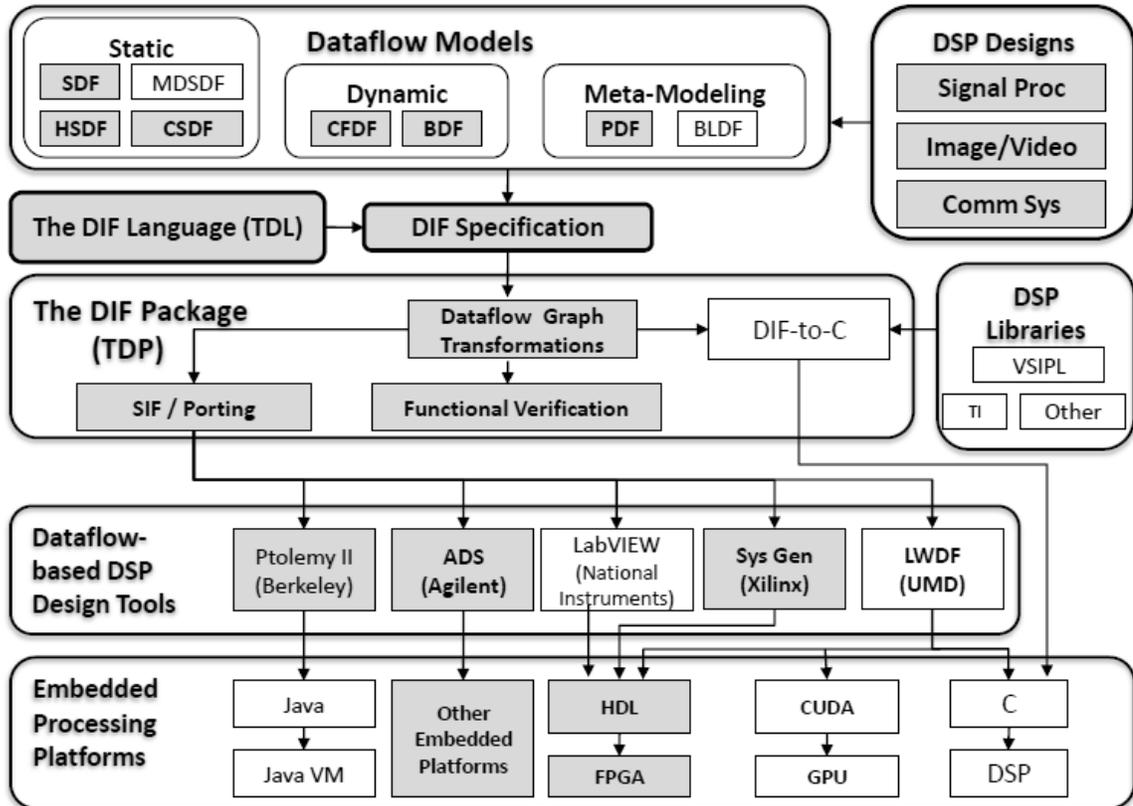


Figure 2.1: TDP-based design flow.

tions while providing interoperability with other design environments and providing a foundation to develop and apply new dataflow-based tools. Developers benefit from the semantics and tool suite in TDP and internal representations can readily be converted into functional implementations with the DIF-to-C tool [12], which is a code synthesis tool for SDF.

2.3 Functional DIF

Functional DIF [11] is a tool that provides efficient simulation and prototyping of dataflow graph functionality and scheduling techniques. The prototyping process in functional DIF allows the designer to not only verify the top-level operation of

the application, but also complete system functionality down to the actor level. The semantic foundation for functional DIF is *core functional dataflow* (CFDF), which allows the designer to specify deterministic, dynamic, dataflow applications. In CFDF, each actor $a \in V$ has a set of *modes*, M_a , that it can execute; an *enable* function to determine if an actor can fire at a given point in time; and an *invoke* function to perform the computation associated with a given mode of actor operation.

At any point of execution, the consumption rate of a CFDF actor input port can be said to be *satisfied* if the number of tokens in the corresponding input FFO is at least equal to the consumption rate of the actor port for the current actor mode. A similar characteristic can be applied to the production rate (in terms of having sufficient empty space in the output buffer) of a CFDF actor output port at any point of the application execution. More details about the enable and invoke functions of an actor can be found in [13].

2.3.1 Targeted DIF (TDIF)

The NT-SIM co-simulator uses TDIF to model applications at the node level. TDIF extends the capabilities of the DIF framework with plug-ins that focus on efficient representations onto embedded platforms [14]. Since it is based on CFDF, TDIF is a flexible dataflow model that supports static and data-dependent dataflow rates, which allows for different production and consumption rates across different actor modes. If each actor only has one mode or if the consumption and production

rates do not vary across modes, then *synchronous* dataflow [9] behavior results and static scheduling techniques can be applied. If all actors in a dataflow graph have SDF behavior, then the enable functions need not be employed at run time since static scheduling can be performed for the overall graph. However, for dynamic dataflow models, the enable functions allow for flexible and efficient construction of low-overhead dynamic or quasi-static schedulers [11, 13].

With different dataflow modeling techniques, TDIF focuses on the flexible design of individual actors. The TDIF language describes high level, platform-independent specifications for the interface behavior of dataflow actors using five keywords: *module*, *input*, *output*, *param*, and *mode*. *Module* declares the actor with a label and defines the coding (target) language used to describe actor functionality. *Input* and *output* declare the actor input and output ports along with the associated port names and token types, respectively. *Param* declares actor parameters along with parameter names and types. *Mode* defines each of the set of CFDF modes associated with an actor. The TDIF compiler then takes the overall actor interface specification and generates an application programming interface (API) in the target language. These generated APIs can take the form of header files in the target language code module. Currently, the TDIF compiler supports C, CUDA, and Verilog as target languages.

After actor compilation using TDIF, an orthogonal compilation process through the DIF framework can take place for the application dataflow graph [14]. Separation of actor and dataflow graph specifications using DIF and TDIF allows designers to focus on actor design using TDIF and overall application or subsystem func-

tionality using DIF. This separation of node and subsystem-level design enhances the agility and efficiency of the system-level design process [15]. The DIF- and TDIF-based design components can be combined using the TDIF synthesis engine, where scheduling results can interact with the dataflow graph and its actors. The top-level implementation is generated by the TDIF synthesis engine. This initializes the *operational contexts* of the actors and their communication channels. An operational context contains an *execution context (EC)*, which encapsulates actor parameters and state variables and a *topological context (TC)*, which encapsulates incident ports [14].

The TDIF environment and design flow are illustrated in Fig. 2.2. By following the design methodology supported by the TDIF and DIF frameworks, application designers can experiment with different scheduling techniques for different platforms in a heterogeneous environment.

2.3.2 Lightweight Dataflow Environment (LIDE)

The LIDE environment is similar to the TDIF environment in that it is also based on the CFDF model of computation. However, TDIF is designed to support extensive automated software synthesis capabilities, whereas the focus on LIDE is to enable efficient experimentation with and integration of different kinds of dataflow modeling techniques with minimal constraints on specialized design tools and libraries. This allows for easy integration of LIDE with existing application design frameworks [16]. In this thesis, we use LIDE to model and implement dataflow-based

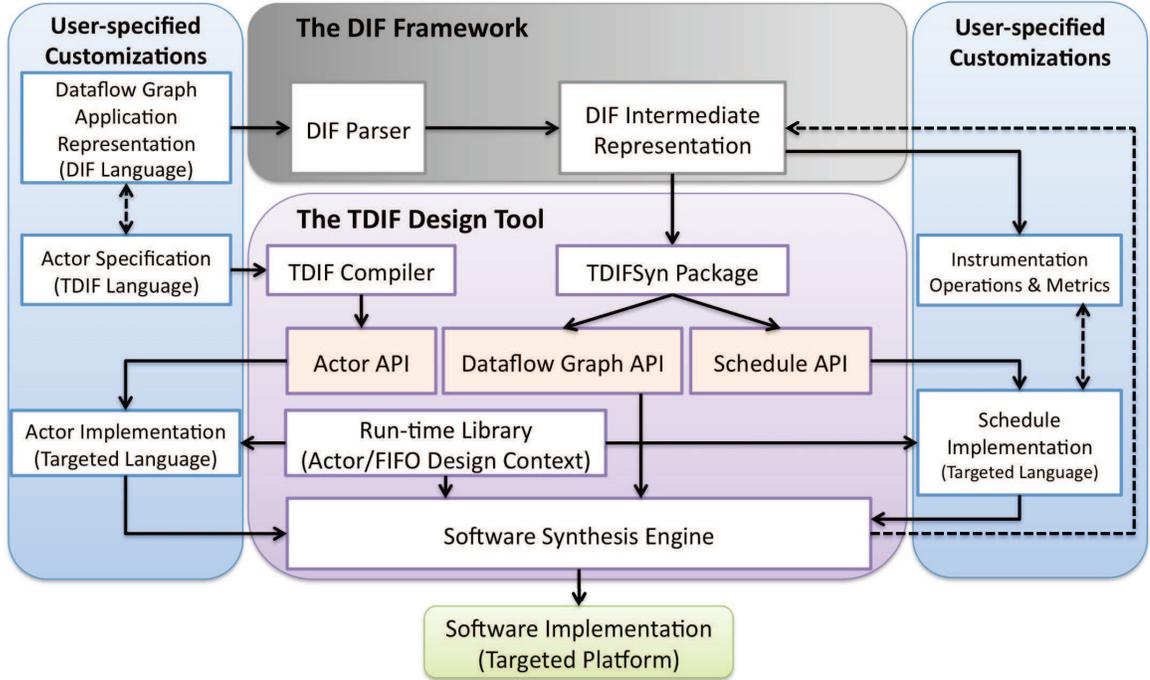


Figure 2.2: TDIF-based design flow.

applications using C.

LIDE is simpler than TDIF. Each LIDE actor is comprised of four main functions: the *new*, *enable*, *invoke*, and *terminate* functions. The *new* function defines the inputs, outputs and parameters of an actor. The purpose of the *enable* and *invoke* functions in LIDE is analogous to their purpose in TDIF. The *terminate* function clears memory usage when an actor instance is no longer needed.

2.4 Network Simulator

The *Network Simulator* (ns-2) is a discrete event simulation tool for networking research with support for the simulation of TCP, routing, and multicast protocols over wired and wireless networks. The simulator is an object-oriented tool based on C++ for protocol implementation and *Object Tcl* (OTcl) for setting up net-

work conditions in simulation. The ns-2 tool incorporates *Tcl with classes* (TclCL) in order to allow objects and variables to exist in a split-language programming environment [8]. This allows designers to work in a reconfigurable simulation environment that separates the simulation primitives in C++ from the simulation design in OTcl [17].

To incorporate end system behaviors, ns-2 contains an emulation feature, *NSE*, which allows the ability to introduce the network simulator into a live network by calling a real-time scheduler to link event execution in simulations to real-time events [8]. This allows for evaluation of both end system and network element behavior. End system components are exposed to packet dynamics that can be hard to reproduce in a live network, but traceable in an emulation environment. Network behavior can be evaluated in relation to end system traffic generation. This allows ns-2 to be used in identifying adverse network element behaviors prior to live network testing [17].

To interface NSE with applications in our NT-SIM environment, *tap agents* and *network objects* are used to pass network data between NSE and TDIF. Tap agents attach network data into simulated packets. Each tap agent is related to a network object, which provides entry and exit points for receiving and sending live network packet data, respectively. There are different types of network objects, depending on the specified network protocol. The current NSE version supports the Pcap/BPF, raw IP, and UDP protocols [8]. For the case studies presented in this paper, we adopt the UDP protocol for communication between nodes.

Chapter 3

Related Work

In this chapter, we discuss related prior work and how the contribution of this thesis helps to advance the state of the art in simulation tools for networked signal processing systems.

The distinguishing characteristic of the proposed NT-SIM and NL-SIM co-simulators is their basis in dataflow concepts for the node-level behavior, and capability of handling network conditions to model and simulate interactions between node-level dataflow graph. This stands in contrast to execution-sequence based co-simulators, which lack dataflow semantics or are restricted to static schedules throughout the network. It also stands in contrast to hybrid system simulators (e.g., see [18]), which focus on interactions between continuous time and discrete time dynamics.

NMLab is a co-simulation framework between MATLAB and ns-2 [19]. NMLab is able to simulate both the application and network by adding network modeling features to MATLAB and automating simulated network communications below the application level of the Open Systems Interconnection reference model (OSI model). Network topologies are constructed using ns-2 and the communication between MATLAB and ns-2 is established using stream sockets connecting with the Tcl interface of ns-2. Scheduling between the two simulators is controlled by

ns-2's scheduler, which synchronizes the simulation time between the application in MATLAB and network in ns-2.

NMLab is able to provide additional functionality to network simulation by forwarding Tcl commands to NMLab classes specified by the designer in ns-2 [19]. In contrast, NT-SIM and NL-SIM utilize the emulation capability of ns-2 to avoid the creation of a new agent for communicating with a specific language, which then allows for heterogeneous system design using different languages. This independence is important for working on heterogeneous platforms — for example, C, CUDA, and Verilog could be used together for microcontroller-, GPU-, and FPGA-based nodes, respectively, in a networked signal processing or data mining simulation. This also allows a real-time simulation of asynchronous events and allows the designer to observe data collisions and node time-outs. In addition, NMLab does not enforce the use of dataflow paradigms for network end nodes, while NT-SIM and NL-SIM enforce the use of dataflow principles with the respective TDIF and LIDE environments, which in turn enables dataflow-based scheduling techniques and other useful methods for analysis and optimization.

PiccSIM is a co-simulator with a graphical user interface that also integrates MATLAB/Simulink and ns-2 [20]. The simulator is specifically designed for networked control systems, where the control application is modeled using MATLAB/Simulink and the network simulation is performed using ns-2. The two simulation environments communicate by passing TCP packets containing XML control messages for ns-2 and UDP packets containing simulation and synchronization data for Simulink. Automatic code generation for the control system and network is

provided [20].

Similar to NMLab, PiccSIM requires the creation of a new agent for the specific purpose of facilitating communication between ns-2 and Simulink. While PiccSIM has the ability to act as a real-time co-simulation framework for control subsystems and simulated networks, questions can be raised about the accuracy of simulating asynchronous distributed systems [21]. Also in contrast to the specialized support for model-based design in MATLAB/Simulink, dataflow capabilities provided by TDIF and LIDE in NT-SIM and NL-SIM, respectively, offer a collection of different types of dataflow modeling styles, and scheduling techniques [14]. In addition, node-level scheduling in PiccSIM only supports single tasking or preemptive multi-tasking, whereas NT-SIM and NL-SIM allow for more flexible scheduling onto single- or multi-processor platforms. Both NT-SIM and NL-SIM also allow designers to experiment with a variety of dataflow graph analysis and optimization techniques when mapping application subsystems onto network nodes.

SystemC-NS-2 uses the simulation environments of SystemC, a C++ class library used to create system models at different abstraction levels, and ns-2 [22]. Both are event-driven simulators with SystemC events tied to hardware-like entities and ns-2 events associated with asynchronous changes on communication channels. The two environments are used to produce capabilities for hardware-network co-simulation. In ns-2, the `ns_sc` agent is used to implement a gateway between ns-2 and SystemC. In SystemC, the new port types `ns_in` and `ns_out` are used to send and receive packets from ns-2 objects [23].

The SystemC-ns-2 co-simulator is similar in some ways to the NT-SIM and

NL-SIM co-simulators. However, SystemC-ns-2 is primarily focused on hardware-oriented network co-simulation, while NT-SIM and NL-SIM are focused on integrating dataflow-based, node-level modeling with network-level simulations. Although SystemC-ns-2 can employ dataflow design techniques, NT-SIM and NL-SIM rigorously enforce such techniques, and are integrated with advanced libraries of dataflow modeling, scheduling, and synthesis techniques geared towards implementing signal processing applications on heterogeneous platforms (e.g., see [14]).

Part of the work presented in this thesis was presented in preliminary form in [24].

Chapter 4

Networked Dataflow Application Simulation

In this chapter, we present our proposed methodology for modeling applications using integrated network- and dataflow-based co-simulators.

4.1 NT-SIM Application Design Framework

NT-SIM is a co-simulation environment that supports design and implementation of networked signal processing and data mining applications on heterogeneous platforms. This is done by simulation of end system behavior using TDIF and network events using NSE. Fig. 4.1 illustrates the execution order and interactions among components in the NT-SIM framework. Application behavior is specified based on dataflow modeling principles using the TDIF framework. To interface with the end system dataflow simulation and traffic generation for the network, the network behavior and protocols used by the nodes are defined by the OTcl script, and simulated by the NSE framework.

To send and receive information between NSE and TDIF, special dataflow actors called *interface actors (IAs)* are used. Unlike conventional dataflow actors, which represent functional components from an application specification, IAs are responsible for traffic generation from TDIF-based modeling subsystems, traffic injection into the NSE framework, and time synchronization between the cooperating

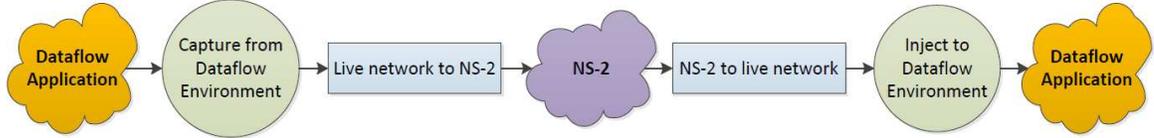


Figure 4.1: Specifying the interaction between dataflow applications and network simulations in NT-SIM.

TDIF- and NSE-based simulation environments. A collection of IAs in a TDIF-based dataflow subsystem effectively makes the subsystem appear as a single node within an enclosing ns-2 network topology. This single node has a number of communication points correlating with the number of IAs.

Simple examples of IAs are send and receive actors based on a client-server relationship using the User Datagram Protocol (UDP). These send and receive actors configure the address and port parameters for the datagram sockets before passing and receiving data from the NSE framework, respectively. A simple example of two-step addition taking place at two different nodes represented by two different TDIF subsystems is shown in Fig. 4.2.

The NT-SIM architecture is designed to preserve dataflow principles provided by the TDIF environment throughout all of the TDIF-based subsystems, including the interactions at all of the subsystem interfaces in the IAs. The responsibility of distributing the actors to network graph nodes lies with the designer. In the NT-SIM framework, the system is developed in a hierarchical manner with TDIF defining actor design, DIF specifying the dataflow graph design, and ns-2 defining the network graph design. Edges in the dataflow graph design act as bridges between actors. IAs act as bridges between dataflow graph subsystems that are distributed among the different network nodes. In NT-SIM, each of the dataflow subsystems

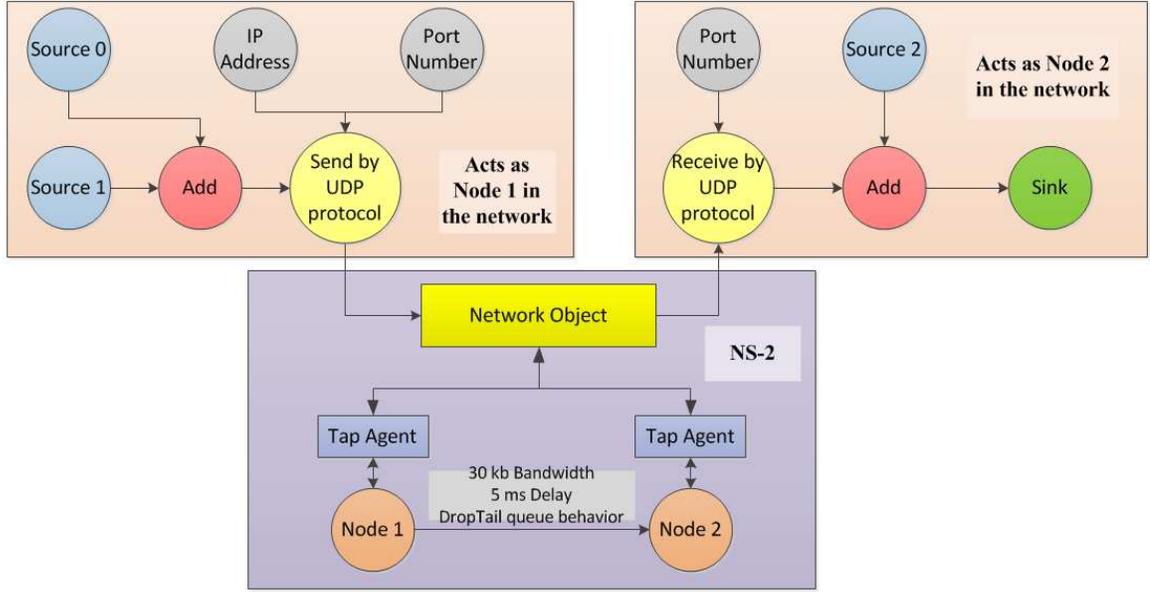


Figure 4.2: A simple example of an NT-SIM networked application system model. In this model, two different nodes perform the addition operation. IAs passing information from and to the Network Object block in the ns-2 subsystem are shown as yellow actors.

can be suspended as they wait for data and resumed arbitrarily while the network is being simulated. This allows for simulation of complex and tightly-coupled feedback patterns in the network. Using this framework, designers can model and simulate their applications using a hierarchical, modular process.

4.1.1 NL-SIM Application Design Framework

Similar to NT-SIM, NL-SIM is a co-simulator that integrates ns-2 with an existing dataflow environment. NL-SIM uses the LIDE framework to specify application behavior. IAs to receive and send data are written in the LIDE design environment to provide equivalent functionality as those in the NT-SIM framework. Unlike NT-SIM, which uses DIF to specify dataflow subsystems, NL-SIM uses a LIDE-C driver function since the LIDE environment has minimal dependencies with

the DIF framework. Like the NT-SIM environment, ns-2 is responsible for specifying network topology and behavior in NL-SIM.

Chapter 5

Case Study: Image Registration Sensor Network

In this chapter, we demonstrate the utility of NT-SIM with a case study of simulating a visual sensor network designed to perform image registration on different views of the same object. This case study is motivated by the rapidly developing field of distributed sensing and its application to areas such as layered sensing, surveillance, videoconferencing, and dynamic data driven adaptive systems (DDDAS) [25, 26, 27].

Instead of gaining knowledge about the environment through a small number of expensive cameras, multiple low-cost cameras can be utilized to provide more complete pictures for challenging, high-level vision tasks such as image registration or tracking [28]. This requires the cameras to be networked together, and to perform collaboration tasks among themselves to optimize key metrics, such as real-time performance, power consumption, and image processing accuracy. Such metrics generally depend on node-network interactions, and thus conventional simulation methods, which consider only network and node characteristics in isolation, are not sufficient. NT-SIM is able to assist in the design of such distributed sensing systems by providing the designer with integrated capabilities to simulate algorithms and applications at the network and node levels.

5.1 Distributed Vision Sensor Systems

Visual sensor networks (VSNs) are comprised of groups of networked visual sensors with image capture, computation, and wireless communication capabilities. To maximize the effectiveness of a VSN, collaboration among the sensors can take place with the exchange or fusing of visual information from similar or different perspectives of an area [28]. This allows the information to be used in tracking, panoramas, and registration.

The scale-invariant feature transform (SIFT) [29] is an algorithm that can be used to fuse together images from multiple cameras that are observing the same object. SIFT uses the difference of Gaussian (DoG) approach to detect feature keypoints at different visual scales. To highlight strong features in the images, the eigenvalues of the Hessian matrix of the image are used to highlight reliable features to use. Results can improve with random sample consensus (RANSAC), which removes outliers and erroneous features detected by the algorithm. Fig. 5.1 shows a dataflow graph model of the SIFT algorithm. Here, the SIFT algorithm is used to register two images with different views of the same object.

Each sensor node in a VSN has to fulfill application requirements while running under constraints involving memory, performance, data rates, and energy [30]. By distributing actors appropriately across the network, more processing-intensive tasks can be performed at one or more stationary systems that are connected to power sources, while simpler tasks are handled by the sensor nodes. This allows energy on the sensor nodes to be conserved while the computationally-intensive task of image

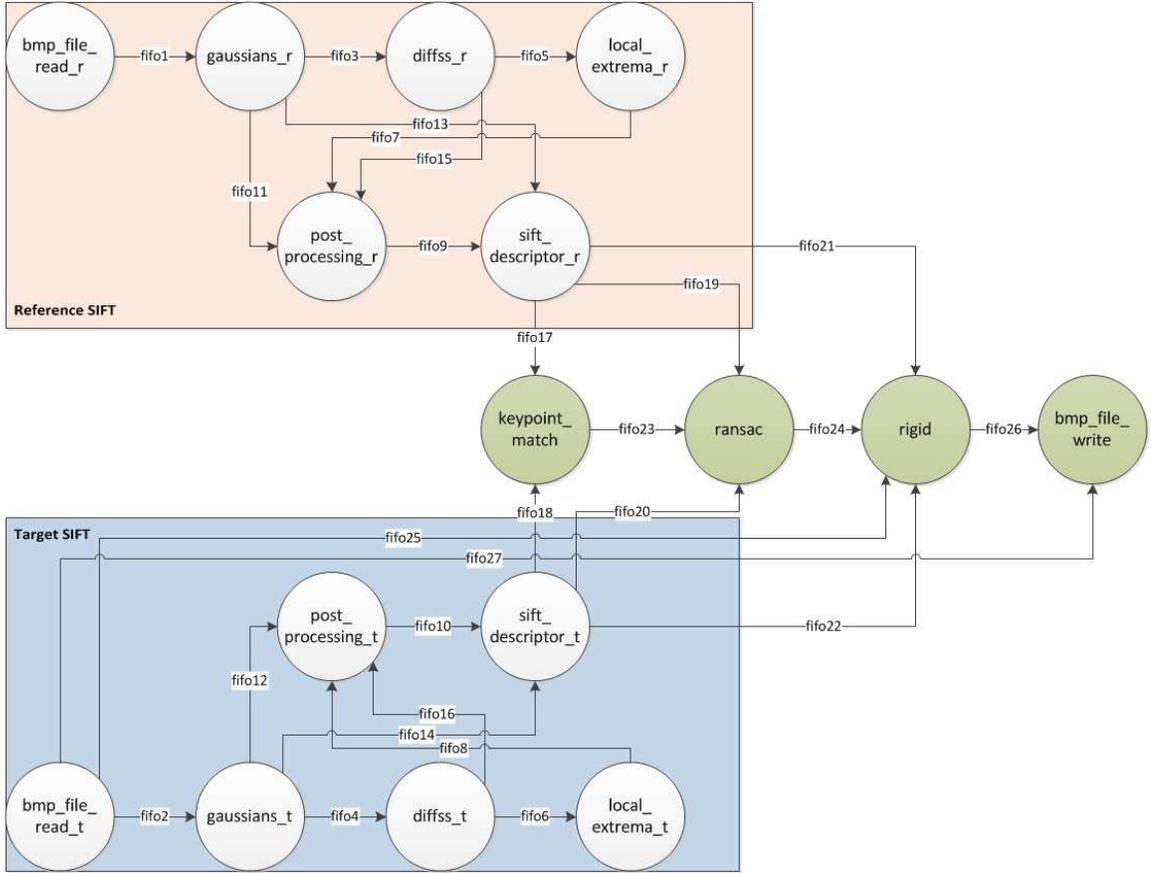


Figure 5.1: A dataflow graph model of SIFT-based feature detection and image registration.

registration is carried out, and also helps to improve the performance of image registration by allowing use of more powerful (less power constrained) platforms for the registration tasks.

In our case study on a SIFT VSN, we experiment with this approach of heterogeneous computing and distribution-based optimization of energy and performance for the SIFT application in a VSN. This experimentation is carried out through mapping of the dataflow graphs for distributed signal processing onto separate network nodes, configuration of IAs in TDIF for appropriate communication among the nodes, and simulation using NT-SIM.

5.2 Actor Design

Each of the actors in the SIFT algorithm is modeled using the TDIF environment. For this purpose, the SIFT algorithm is broken into smaller procedural units to be modeled with actors. At this level of NT-SIM, the actors are not assigned to any particular nodes in a network. The focus at the actor design level of NT-SIM is to create actors that are represented by the TDIF language. In this phase of the design process, designers specify the target language of each actor, along with the inputs, outputs, required parameters, and possible execution modes for the actor. This is carried out for each actor in Fig. 5.1 to write the corresponding TDIF files for the actors. Listing 5.1 shows the TDIF file for the SIFT descriptor actor, which passes the SIFT descriptor to the keypoint matching, RANSAC, and rigid transformation actors. The SIFT descriptor actor represented in Listing 5.1 is specified as a CUDA-targeted actor for GPU-based implementation.

Listing 5.1: TDIF code for the SIFT descriptor actor.

```
1 module CUDA sift_descriptor_r
2
3 output output1 sift_token
4 output output2 sift_token
5 output output3 sift_token
6
7 input input1 oframes
8 input input2 gss
9
10 mode init
11 mode exe
```

To partition the dataflow graph represented in Fig. 5.1 across multiple network nodes, the designer can identify desired network interface locations on the original SIFT application graph (Fig. 5.1), and then specify send and receive IAs at

these graph locations with the TDIF language. This will then partition the graph, and connect the desired points in the graphs with NSE to achieve the appropriate connections with the simulated communication network.

As an example, Listing 5.2 shows TDIF code for sending an image from the actor representing the capture of a target image to the network simulated by ns-2. For simplicity and clarity in the illustration, we design the network to follow the UDP protocol. As a result, the image-sending actor represented by Listing 5.2 takes in the address and port number as character-string parameters, and these parameters are employed by the actor in addition to any inputs coming from other actors in the enclosing dataflow graph subsystem.

Listing 5.2: TDIF code for sending an image to NSE via the UDP protocol.

```
1 module C send_udp_sift_t_img
2
3 input input_image image_token*
4
5 param send_addr char*
6 param send_port char*
7
8 mode init
9 mode send
```

5.3 Actor Separation at the Node Level

In NT-SIM, the application that runs on each network node is represented by a specification in the DIF language. To optimize the energy and performance of the SIFT VSN, actors are split onto different network nodes depending on their roles in the overall application graph. This results in multiple dataflow graph subsystems with each subsystem corresponding to a single network node. Each of these subsystems can be specified using a DIF file that defines the actors as vertices and the connections between them as edges in the associated dataflow graph.

In this case study, the actors are distributed across network nodes depending

on whether they perform feature detection or image registration. Fig. 5.2 shows the dataflow graph subsystem for feature detection, and Fig. 5.3 shows the subsystem for the registration of the reference and target image, given the SIFT descriptors of the features located in both images.

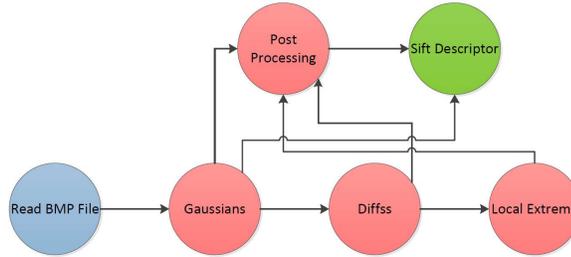


Figure 5.2: Dataflow graph subsystem for SIFT feature detection.

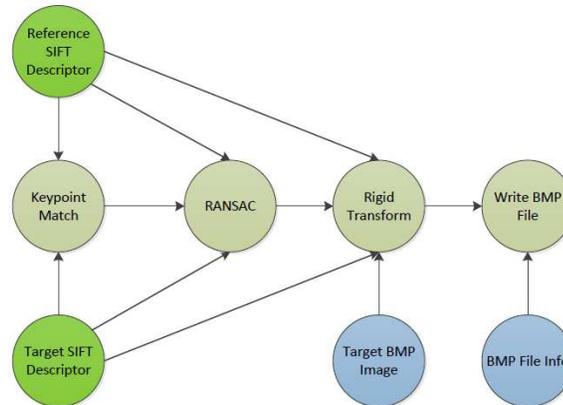


Figure 5.3: Subsystem for image registration after SIFT feature detection.

Currently, the designer creates the test and schedule files for each of the dataflow graph systems. However, the process of creating these files can be automated, and we will explore such automation in our future development of the TDIF synthesis engine. Our current version of NT-SIM systematically integrates the designer-provided tests and schedules into the overall network simulation, and automates the execution of this simulation across the entire network. Thus, NT-SIM bridges the gap between network- and dataflow-graph-level simulation in networked signal processing systems, and provides novel capabilities into which existing and

newly developed dataflow scheduling techniques can be integrated to further enhance simulation automation and design space exploration.

5.4 Network Creation

When using NT-SIM, the designer creates a Tcl script that models the network topology on NSE to simulate the network. In order to use NSE on ns-2, the `RealTime` scheduler has to be used with the simulator. Nodes are declared along with the network objects and agents. When using the UDP protocol, each of the network objects must declare the IP address and port number in the script. These network objects are attached to their corresponding agents. Afterwards, the connections between nodes can be defined, along with the bandwidth, delay, and queue behavior for each connection.

Each agent is attached to a node. If the nodes share a common link, then the agents are also connected. Afterwards, NSE can be run. Fig. 5.4 illustrates the network topology used in our SIFT VSN case study. Listings 5.3 shows part of the Tcl script used to represent the network connections in Fig. 5.4.

Listing 5.3: An excerpt from the Tcl script used to implement the connection between the reference and registration nodes in the SIFT sensor network simulation.

```
1 # Create RealTime simulator object
2 set ns [new Simulator]
3 $ns use-scheduler RealTime
4
5 # Define network nodes
6 set node0 [$ns node]
7 set node1 [$ns node]
8 set node2 [$ns node]
9 set node3 [$ns node]
10 set node4 [$ns node]
11 set node5 [$ns node]
12 set node6 [$ns node]
13 set node7 [$ns node]
14
15 # Introduce live UDP traffic from SIFT reference through 11000 port
```

```

16 set in_sift_r [new Network/IP/UDP]
17 $in_sift_r open readonly
18 $in_sift_r bind 127.0.0.1 11000
19 set in_sift_r_a [new Agent/Tap]
20 $in_sift_r_a network $in_sift_r
21
22 # Introduce live UDP traffic from SIFT target through 11010 port
23 set in_sift_t [new Network/IP/UDP]
24 $in_sift_t open readonly
25 $in_sift_t bind 127.0.0.1 11010
26 set in_sift_t_a [new Agent/Tap]
27 $in_sift_t_a network $in_sift_t
28
29 # Introduce live UDP traffic from target image through 11020 port
30 set in_sift_t_img [new Network/IP/UDP]
31 $in_sift_t_img open readonly
32 $in_sift_t_img bind 127.0.0.1 11020
33 set in_sift_t_img_a [new Agent/Tap]
34 $in_sift_t_img_a network $in_sift_t_img
35
36 # Introduce live UDP traffic from target BMP info through 11030 port
37 set in_sift_t_bmp [new Network/IP/UDP]
38 $in_sift_t_bmp open readonly
39 $in_sift_t_bmp bind 127.0.0.1 11030
40 set in_sift_t_bmp_a [new Agent/Tap]
41 $in_sift_t_bmp_a network $in_sift_t_bmp
42
43 # Define UDP network object to output SIFT reference through 12000 port
44 set out_sift_r [new Network/IP/UDP]
45 $out_sift_r open writeonly
46 $out_sift_r connect 127.0.0.1 12000
47 set out_sift_r_a [new Agent/Tap]
48 $out_sift_r_a network $out_sift_r
49
50 # Define UDP network object to output SIFT target through 12010 port
51 set out_sift_t [new Network/IP/UDP]
52 $out_sift_t open writeonly
53 $out_sift_t connect 127.0.0.1 12010
54 set out_sift_t_a [new Agent/Tap]
55 $out_sift_t_a network $out_sift_t
56
57 # Define UDP network object to output target image through 12020 port
58 set out_sift_t_img [new Network/IP/UDP]
59 $out_sift_t_img open writeonly
60 $out_sift_t_img connect 127.0.0.1 12020
61 set out_sift_t_img_a [new Agent/Tap]
62 $out_sift_t_img_a network $out_sift_t_img

```

```

63
64 # Define UDP network object to output SIFT target through 12030 port
65 set out_sift_t_bmp [new Network/IP/UDP]
66 $out_sift_t_bmp open writeonly
67 $out_sift_t_bmp connect 127.0.0.1 12030
68 set out_sift_t_bmp_a [new Agent/Tap]
69 $out_sift_t_bmp_a network $out_sift_t_bmp
70
71 # Connect the nodes and agents
72 $ns duplex-link $node0 $node4 30kb 5ms DropTail
73 $ns duplex-link $node1 $node5 30kb 5ms DropTail
74 $ns duplex-link $node2 $node6 2048kb 5ms DropTail
75 $ns duplex-link $node3 $node7 30kb 5ms DropTail
76 $ns attach-agent $node0 $in_sift_r_a
77 $ns attach-agent $node1 $in_sift_t_a
78 $ns attach-agent $node2 $in_sift_t_img_a
79 $ns attach-agent $node3 $in_sift_t_bmp_a
80 $ns attach-agent $node4 $out_sift_r_a
81 $ns attach-agent $node5 $out_sift_t_a
82 $ns attach-agent $node6 $out_sift_t_img_a
83 $ns attach-agent $node7 $out_sift_t_bmp_a
84 $ns connect $in_sift_r_a $out_sift_r_a
85 $ns connect $in_sift_t_a $out_sift_t_a
86 $ns connect $in_sift_t_img_a $out_sift_t_img_a
87 $ns connect $in_sift_t_bmp_a $out_sift_t_bmp_a
88
89 # Run the simulation
90 $ns run

```

5.5 Simulation of the Distributed System

After the actors, dataflow graph subsystems (the portions of the dataflow graph that are mapped onto individual network nodes), and the network have been specified, the overall system can be simulated using NT-SIM. The Tcl script for the network is run using NSE. This allows network connections to be made between the TDIF and ns-2 environments. Separate test and DIF files are required for each VSN node. After the executables have been generated for each VSN node, they can be run — concurrently with simulation of the resulting network traffic — to send and receive data to and from NSE, respectively.

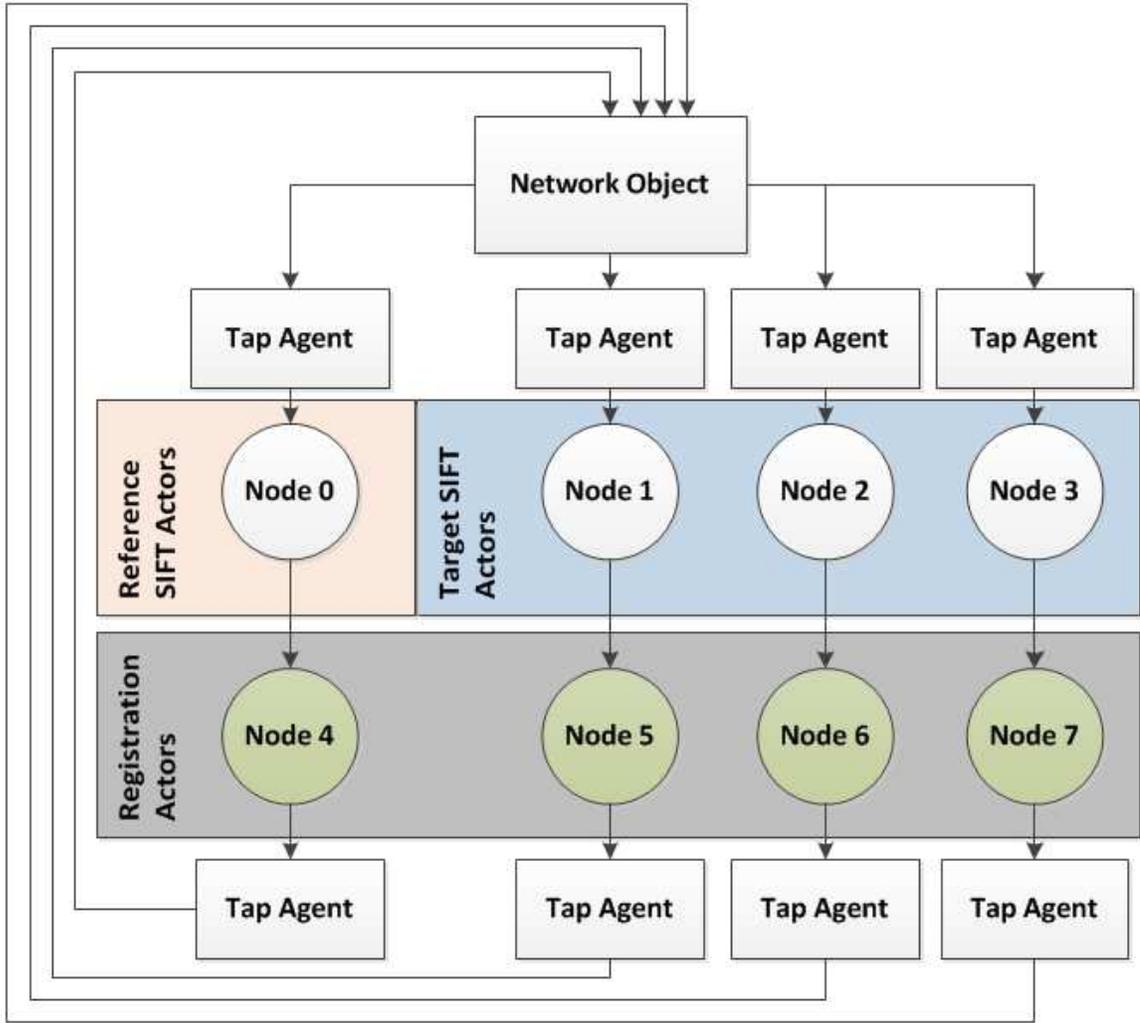


Figure 5.4: The topology represented by the Tcl script for the SIFT sensor network.

The output of the dataflow graph subsystem responsible for image registration can be used for testing by comparing against known results. Fig. 5.5 illustrates the overall simulated network for our case study on a SIFT VSN with two visual sensors performing feature detection on captured images and a main computing node that performs the image registration of the target and reference images. This diagram can be viewed as an interconnection of the dataflow graph subsystems involved in the distributed and heterogeneous signal processing configuration for the targeted VSN application.

The SIFT sensor network is simulated on a 3GHz PC with two Intel Xeon

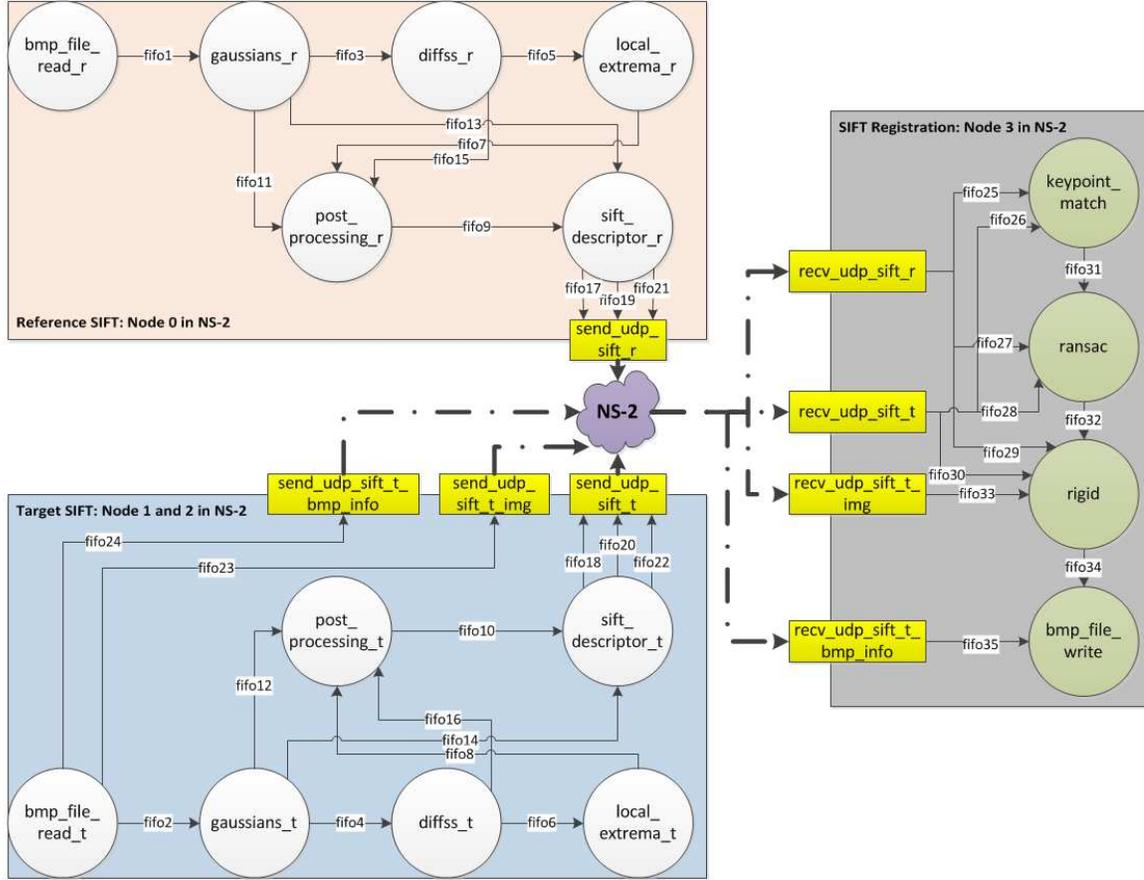


Figure 5.5: Graphical representation of the simulated SIFT sensor network.

CPUs, 3GB RAM, and an NVIDIA GTX260 GPU. The `gcc` version 3.4.4 and `nvcc` version 3.2 compilers are used in the back end of the implementation process.

The functional accuracy of NT-SIM was verified through simulation of the SIFT VSN case study. End systems (network nodes) representing reference and target image sensors that can perform feature detection were supplied with only the reference and target image shown in Fig. 5.6 and Fig. 5.7, respectively.

NSE was run to provide the network across end systems. The node responsible for image registration was run after NSE to start listening for outputs from the NSE network represented by the Tcl script. Afterwards, the nodes responsible for feature detection of the reference and target images were run. The simulation completed



Figure 5.6: Reference image used for SIFT feature detection.

with the output of the registered image shown in Fig. 5.8. Functional accuracy was validated by the match between the produced, registered image and a ground-truth, registered image provided by the simulation of the single-node SIFT algorithm shown in Fig. 5.1. Although the current implementation of NT-SIM has only been tested using a local machine, it can readily be extended to exploit networks of multiple machines for simulation — e.g., by exploiting parallelism within and across the dataflow graphs within a simulated system.



Figure 5.7: Target image used for SIFT feature detection.



Figure 5.8: Resulting registered image from SIFT VSN case study using NT-SIM.

Chapter 6

Case Study: Adaptive Stream Mining

We demonstrate the utility of NL-SIM with a case study of face detection using multiple classifiers in a distributed network. This case study is motivated by the increasing relevance of embedded systems for adaptive stream mining (ASM), where machine learning is integrated deeply not only with performance constraints, but also with resource constraints [31, 32, 33].

Instead of relying on a powerful system with a strong classifier, ASM systems may use multiple, weak classifiers that can be reconfigured to different topologies or parameters to extract more meaningful data in runtime- or memory-constrained systems [34, 35, 36]. Such ASM systems require implementations that can systematically switch among configurations depending on the input data or the performance constraints. Such an approach requires subsystems that can communicate efficiently with other subsystems to coordinate responsibilities. By monitoring data and performance using NL-SIM, designers can create ASM systems that address various data and performance constraints with a given set of classifiers.

6.1 Support Vector Machines

Support vector machines (SVM) are supervised learning models that can be used for classification purposes. A trained SVM classification model would take input data and calculate a value, which can then be thresholded to determine the class of the data. Conceptually, an SVM model is a representation of the training examples as hyperplanes in space with different classes separated by the widest gap allowed in the mapped space. The examples that are used to construct this

maximum margin are known as support vectors (SV). The boundary formed by the SVs determine the classification of the input data. Fig. 6.1 shows a simple example of a trained, linear SVM classifier used to separate data between two classes.

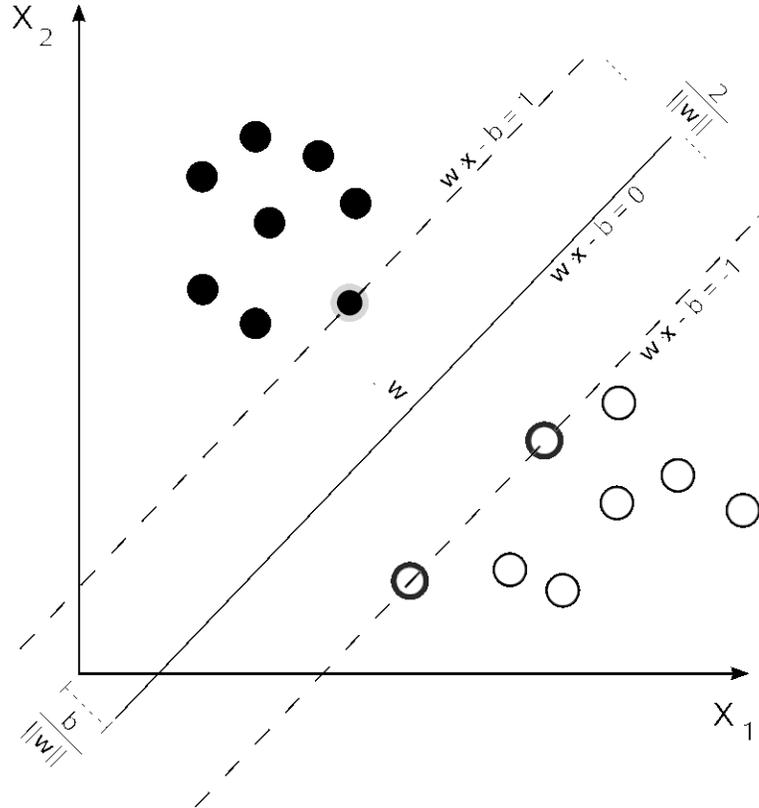


Figure 6.1: A simple example of classification using a linear support vector machine. Points on the boundary represent the support vectors.

Nonlinear classification using SVMs can achieve good performance for the task of face detection. One of the most popular kernels is the Gaussian radial basis function (RBF), defined by $k(x_i, x_{SV}) = \exp(-\gamma \|x_i - x_{SV}\|^2)$, where x represents the data point and γ is a parameter that can be configured. Along with the *gamma* parameter in the Gaussian RBF kernel, an SVM model can be modified by training on a different box constraint, which is used in the training process. This box constraint, C , is the soft margin that controls the margin by weighting the error between the model and the data.

The process of cross-validation can be used to determine optimal parameter

values for each SVM based on the needs of the application. A part of the training data is withheld while the classifier is trained on the remaining training data. By testing on the withheld set of data, the designer can simulate the performance of the classifier with chosen parameter values. By checking the false positive and negative rates, the designer can estimate the best classifier to use for cases where a specific type of accuracy is important. The number of SVs for each classifier directly affects the runtime of the classifier.

In this case study, we experiment with three SVM classifiers designed with different performance goals: high accuracy, low runtime, and low false positive rates. This experimentation is carried out through mapping of the dataflow graphs of SVM classification using the different classifiers onto separate subsystems in the ACM actor, selection of the classifier to use based on situational goals, and simulation using the NL-SIM environment.

6.2 Actor-Level Design

The actors in our experiments are modeled using the LIDE-C environment. At this level of application design in NL-SIM, each of the actors are not assigned to any particular subsystem. Instead, designers specify the inputs, outputs, parameters, and execution modes for the actor `context` in the `new` function. In the face detection application, the main actor for the subsystems is the trained SVM classifier. Listing 6.1 shows the actor context for the actor for SVM classification. Listing 6.2 shows the corresponding `new` function.

Listing 6.1: LIDE context for the SVM classification actor.

```

1 struct _lide_c_test_svm_context_struct {
2 #include "lide_c_actor_context_type_common.h"
3
4     /* local variables and input data*/
5     int num_dims;
6     int num_sv;
```

```

7   float bias;
8   float rbf_sigma;
9   float *data;
10  float *alpha;
11  float **sv;
12  /* input and output ports */
13  lide_c_fifo_pointer input_data;
14  lide_c_fifo_pointer input_svs;
15  lide_c_fifo_pointer input_alphas;
16  lide_c_fifo_pointer output_class;
17  };

```

Listing 6.2: LIDE code for the new function of the SVM classification.

```

1  lide_c_test_svm_context_type *lide_c_test_svm_new(
2      lide_c_fifo_pointer input_data, lide_c_fifo_pointer input_svs,
3      lide_c_fifo_pointer input_alphas, lide_c_fifo_pointer output_class,
4      float bias, float rbf_sigma, int num_svs, int num_dims) {
5      lide_c_test_svm_context_type *context = NULL;
6      int i = 0;
7
8      context = lide_c_util_malloc(sizeof(lide_c_test_svm_context_type));
9      /* Function specification */
10     context->mode = LIDE_C_TEST_SVM_MODE_LOAD;
11     context->enable = (lide_c_actor_enable_function_type)lide_c_test_svm_enable;
12     context->invoke = (lide_c_actor_invoke_function_type)lide_c_test_svm_invoke;
13     /* Constant specification */
14     context->num_dims = num_dims;
15     context->num_sv = num_svs;
16     context->bias = bias;
17     context->rbf_sigma = rbf_sigma;
18     /* Data specification */
19     context->data = lide_c_util_malloc(sizeof(float) * context->num_dims);
20     context->alpha = lide_c_util_malloc(sizeof(float) * context->num_sv);
21     context->sv = lide_c_util_malloc(sizeof(float*) * context->num_sv);
22     for (i = 0; i < context->num_sv; i++) {
23         *(context->sv + i) = lide_c_util_malloc(
24             sizeof(float) * context->num_dims);
25     }
26     /* Files specification */
27     context->input_data = input_data;
28     context->input_svs = input_svs;
29     context->input_alphas = input_alphas;
30     context->output_class = output_class;
31     return context;
32 }

```

To decide the actor placement at each of the network nodes, the designer can connect the send and receive IAs at the beginning and end of each subsystem dataflow graph. Similar to NT-SIM, these actors can specify desired network interface locations on the face detection application. This partitions the graph into nodes that can be connected with NSE to simulate connections with a communication network. The corresponding LIDE code for the sending actor is represented by Listing 6.3.

Listing 6.3: LIDE context for sending image data via the UDP protocol.

```

1 struct _lide_c_send_udp_context_struct {
2 #include "lide_c_actor_context_type_common.h"
3     char *send_addr;
4     int send_port;
5     lide_c_fifo_pointer in_mode;
6     lide_c_fifo_pointer in_data;
7 };

```

6.3 Subsystem Level Design

In NL-SIM, the face detection application that runs on each network node is represented by a driver function in the LIDE-C language. To optimize the discrimination of the face detection network, each network node contains an SVM classifier with different performance characteristics: high accuracy, low runtime, or low false positive rate. There is one additional node to read in the input face images and send them to the each SVM classifier, depending on the requested operating mode.

The dataflow graphs for each subsystem in this application are similar to one another. The main difference is in the SVM parameters and SVs to be used to classify the images. Fig. 6.2 shows the dataflow graph subsystem for reading in the images and Fig. 6.3 shows the subsystem for the classification of images.

Currently, the designer creates test files for each of the dataflow graph systems. Each subsystem can only run on a simple scheduler, where each actor fires when

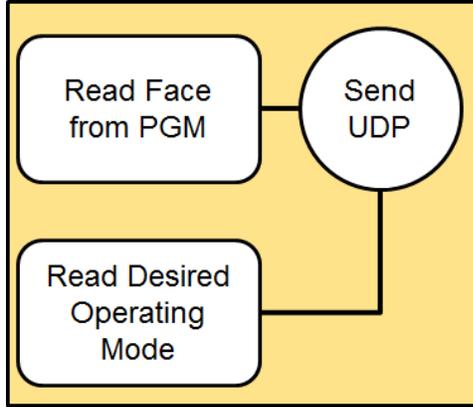


Figure 6.2: Dataflow graph subsystem for reading in face images.

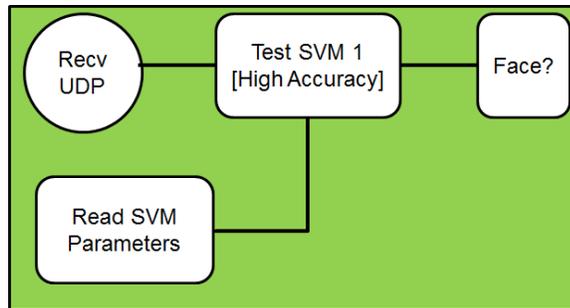


Figure 6.3: Subsystem for SVM classification.

enable conditions are fulfilled. However, future versions that allow different types of schedules to be run in the LIDE environment are being explored. Our current version of NL-SIM integrates the tests into the overall network simulation and automates the execution across the entire network. Thus, NL-SIM provides the same capabilities as NT-SIM in bridging together the gap between network- and dataflow-graph-level simulation in networked signal processing and data mining systems.

6.4 Network Design

Similar to NT-SIM, the designer creates a Tcl script to model the master-slave network topology on NSE. The nodes communicate using the UDP protocol, with each of the network objects declaring an IP address and port number in the script. The connections between the nodes are matched with each other in one-to-

one correspondence similar to that seen in Fig. 5.4.

6.5 Simulation of the Adaptive Stream Mining System

After the actors, dataflow graph subsystems, and the network have been specified, the overall system can be simulated using NL-SIM. The Tcl script for the network is run using NSE. This allows network connections to be made between the LIDE and ns-2 environments. Separate test and LIDE-C driver function files are required for each node. After the executables have been generated for each node in the face detection application, they can be run — concurrently with simulation of the resulting network traffic — to send and receive data to and from NSE, respectively.

The output of the dataflow graph subsystem responsible for image registration can be used for testing by comparing against known results. Fig. 6.4 illustrates the overall simulated network for our case study on a SIFT VSN with two visual sensors performing feature detection on captured images and a main computing node that performs the image registration of the target and reference images. This diagram can be viewed as an interconnection of the dataflow graph subsystems involved in the distributed and heterogeneous signal processing configuration for the targeted VSN application.

The face detection network application is simulated on a 3GHz PC with two Intel Xeon CPUs and 3GB RAM. The `gcc version 3.4.4` compiler is used in the back end of the implementation process.

The functional accuracy of NL-SIM was verified through simulation of a face from the MIT CBCL database [37]. The sample face run through each of the classification subsystems and the resulting classification are shown within Fig. 6.4. Functional accuracy was validated through comparisons with values attained from SVM classification in MATLAB. Although the current implementation of NL-SIM has been tested using a local machine, LIDE capabilities allow NL-SIM to support

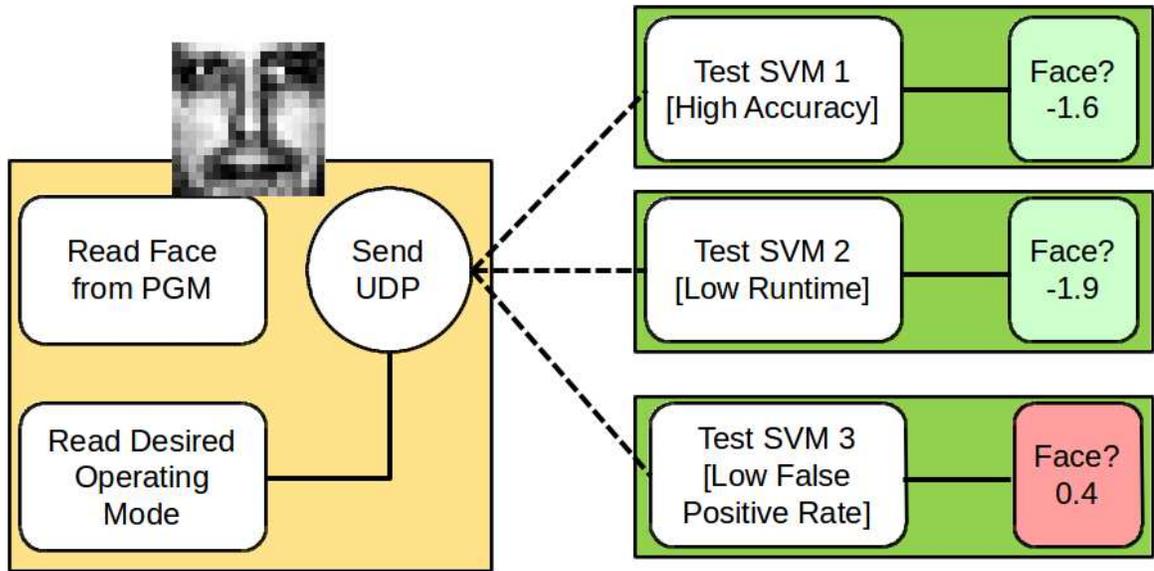


Figure 6.4: Graphical representation of the simulated face detection network.

a network of multiple machines for simulation. Developing and experimenting with such capabilities is a useful direction for future work.

Chapter 7

Conclusions and Future Work

In this thesis, we have presented design methodologies and two tools, called NT-SIM and NL-SIM, for simulating and experimenting with networked signal processing systems. We have shown that the NT-SIM and NL-SIM environments provide designers with a hierarchical, modular process for modeling and experimenting with networked signal processing and data mining systems. Furthermore, NT-SIM also provides a useful target for incorporating additional levels of automation in the design and simulation processes. For example, protocol configurations and associated implementation details can be determined and optimized automatically by incorporating associated IA synthesis capabilities within the TDIF synthesis engine. Building on both co-simulators to develop such new automation and optimization capabilities is an interesting and useful direction for future work.

We have introduced NT-SIM as a co-simulation tool that combines the dataflow methods of TDIF and DIF for actor and dataflow graph design, respectively, and the network simulation capabilities of NSE. We have also introduced NL-SIM as a co-simulation tool that integrates the dataflow methods of LIDE for actor and dataflow graph design with NSE network simulation capabilities. The resulting tools provide useful new capabilities for flexible and accurate simulation of networked signal processing systems. In particular, given the growing use of dataflow methods in design and optimization of signal processing systems, it is important simulate the impact of dataflow techniques in the context of the overall networked environment in which they operate. The techniques and tools introduced in this thesis help to advance the state of the art in this direction.

We have demonstrated that using the NT-SIM and NL-SIM co-simulators, a

designer can simulate complete, networked systems comprised of a distinct application subsystems on each network node with actors modeled using formal dataflow-based representations. The useful features of NT-SIM and NL-SIM include their modular design flow, where actors are designed using the TDIF or LIDE tool, application graphs are modeled in the DIF or LIDE framework, and the network is represented in ns-2.

Useful directions for further development of the co-simulators include automating the separation of an application dataflow graph across a network through the TDIF synthesis engine or LIDE environment, application of instrumentation actors in TDIF and LIDE to encapsulate relevant network performance measurements provided by NSE, and incorporating different network protocols along with promoting reuse of the associated protocol code as TDIF or LIDE actor library components.

Bibliography

- [1] A. Gerasoulis and T. Yang. On the granularity and clustering of directed acyclic task graphs. *IEEE Transactions on Parallel and Distributed Systems*, pages 686–701, June 1993.
- [2] V. Kianzad and S. S. Bhattacharyya. Efficient techniques for clustering and scheduling onto embedded multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 17(7):667–680, July 2006.
- [3] Y. Kwok and I. Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *Journal of the Association for Computing Machinery*, 31(4):406–471, December 1999.
- [4] H. J. Siegel, J. K. Antonio, R. C. Metzger, M. Tan, and Y. A. Li. Heterogeneous computing. In A. Y. Zomaya, editor, *Parallel and Distributed Computing Handbook*. McGraw-Hill, 1996.
- [5] S. Sriram and S. S. Bhattacharyya. *Embedded Multiprocessors: Scheduling and Synchronization*. CRC Press, second edition, 2009.
- [6] L. Wang, H. J. Siegel, and V. Roychowdhury. A genetic-algorithm-based approach for task matching and scheduling in heterogeneous environments. In *Proceedings of the Heterogeneous Computing Workshop*, pages 72–85, April 1996.
- [7] S. S. Bhattacharyya, E. Deprettere, R. Leupers, and J. Takala, editors. *Handbook of Signal Processing Systems*. Springer, 2010.
- [8] K. Fall and K. Varadhan. *The ns Manual (formerly ns Notes and Documentation)*, November 2011.
- [9] E. A. Lee and D. G. Messerschmitt. Synchronous dataflow. *Proceedings of the IEEE*, 75(9):1235–1245, September 1987.
- [10] J. T. Buck and E. A. Lee. Scheduling dynamic dataflow graphs using the token flow model. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, April 1993.
- [11] W. Plishker, N. Sane, M. Kiemb, K. Anand, and S. S. Bhattacharyya. Functional DIF for rapid prototyping. In *Proceedings of the International Symposium on Rapid System Prototyping*, pages 17–23, Monterey, California, June 2008.
- [12] C. Hsu, M. Ko, and S. S. Bhattacharyya. Software synthesis from the dataflow interchange format. In *Proceedings of the International Workshop on Software and Compilers for Embedded Systems*, pages 37–49, Dallas, Texas, September 2005.

- [13] S. S. Bhattacharyya, W. Plishker, N. Sane, C. Shen, and H. Wu. Modeling and optimization of dynamic signal processing in resource-aware sensor networks. In *Proceedings of the Workshop on Resources Aware Sensor and Surveillance Networks in conjunction with IEEE International Conference on Advanced Video and Signal-Based Surveillance*, pages 449–454, Klagenfurt, Austria, August 2011.
- [14] C. Shen, H. Wu, N. Sane, W. Plishker, and S. S. Bhattacharyya. A design tool for efficient mapping of multimedia applications onto heterogeneous platforms. In *Proceedings of the IEEE International Conference on Multimedia and Expo*, Barcelona, Spain, July 2011. 6 pages in online proceedings.
- [15] K. Keutzer, S. Malik, R. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli. System-level design: orthogonalization of concerns and platform-based design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19, December 2000.
- [16] C. Shen, W. Plishker, and S. S. Bhattacharyya. Dataflow-based design and implementation of image processing applications. Technical Report UMIACS-TR-2011-11, Institute for Advanced Computer Studies, University of Maryland at College Park, 2011. <http://drum.lib.umd.edu/handle/1903/11403>.
- [17] S. Bajaj et al. Improving simulation for network research. Technical Report 99-702b, University of Southern California, March 1999.
- [18] A. Chutinan and B. H. Krogh. Computational techniques for hybrid system verification. *IEEE Transactions on Automatic Control*, 48(1):64–75, 2003.
- [19] O. Heimlich, R. Sailer, and L. Budzisz. NMLab: A co-simulation framework for Matlab and ns-2. In *Proceedings of the International Conference on Advances in System Simulation*, pages 152–157, 2010.
- [20] T. Kohtamaki, M. Pohjola, J. Brand, and L. M. Eriksson. PiccSIM toolchain — design, simulation and automatic implementation of wireless networked control systems. In *Proceedings of the International Conference on Networking, Sensing and Control*, pages 49–54, 2009.
- [21] G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair. *Distributed Systems: Concepts and Design*. Addison-Wesley, 2011.
- [22] F. Fummi, G. Perbellini, P. Gallo, M. Poncino, S. Martini, and F. Ricciato. A timing-accurate modeling and simulation environment for networked embedded systems. In *Proceedings of the Design Automation Conference*, pages 42–47, 2003.
- [23] G. Perbellini. SystemC – NS-2 co-simulation using HSN. Technical report, Universita degli Studi di Verona, December 2005.

- [24] S. Won, C. Shen, and S. S. Bhattacharyya. NT-SIM: A co-simulator for networked signal processing applications. In *Proceedings of the European Signal Processing Conference*, Bucharest, Romania, August 2012. To appear.
- [25] M. Bryant, P. Johnson, B. M. Kent, M. Nowak, and S. Rogers. Layered sensing: Its definition, attributes, and guiding principles for AFRL strategic technology development. Technical report, Sensors Directorate, U.S. Air Force Research Laboratory, May 2008.
- [26] A. C. Sankaranarayanan, A. Veeraraghavan, and R. Chellappa. Object detection, tracking and recognition for multiple smart cameras. *Proceedings of the IEEE*, 96(10):1606–1624, October 2008.
- [27] F. Damera. Grid computing and beyond: The context of dynamic data driven applications systems. *Proceedings of the IEEE*, 93(2):692–697, 2005.
- [28] Y. Bai and H. Qi. Feature-based image comparison for semantic neighbor selection in resource-constrained visual sensor networks. *EURASIP Journal on Image and Video Processing*, 2010. doi:10.1155/2010/469563.
- [29] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.
- [30] I. F. Akyildiz, T. Melodia, and K. R. Chowdhury. Wireless multimedia sensor networks: Applications and testbeds. *Proceedings of the IEEE*, 96(10):1588–1605, October 2008.
- [31] M. Shah, J. Hellerstein, S. Chandrasekaran, and M. Franklin. Flux: An adaptive partitioning operator for continuous query systems. In *Proceedings of the International Conference on Data Engineering*, 2003.
- [32] M. Cherniack, H. Balakrishnan, D. Carney, U. Cetintemel, Y. Xing, and S. Zdonik. Scalable distributed stream processing. In *CIDR*, 2003.
- [33] C. Olston, J. Jiang, and J. Widom. Adaptive filters for continuous queries over distributed data streams. In *ACM SIGMOD*, 2003.
- [34] F. Fu, D. Turaga, O. Verscheure, M. van der Schaar, and L. Amini. Configuring competing classifier chains in distributed stream mining systems. *IEEE Journal on Selected Topics in Signal Processing*, 1(4):548–563, December 2007.
- [35] B. Foo, D. S. Turaga, O. Verscheure, M. van der Schaar, and L. Amini. Resource constrained stream mining with classifier tree topologies. *IEEE Signal Processing Letters*, 15:761–764, 2008.
- [36] R. Ducasse, D. S. Turaga, and M. van der Schaar. Ordering of stream mining classifiers. In *Proceedings of the International Conference on Image Processing*, pages 3177–3180, 2010.

[37] CBCL face database #1. <http://cbcl.mit.edu/software-datasets/FaceData2.html>, 2010.