# ABSTRACT

Title of dissertation: DESIGN TOOLS FOR DYNAMIC, DATA-DRIVEN, STREAM MINING SYSTEMS

Kishan Palintha Sudusinghe,
Doctor of Philosophy, 2015

Dissertation directed by: Professor Shuvra S. Bhattacharyya
Department of Electrical and Computer Engineering
and Institute for Advanced Computer Studies

The proliferation of sensing devices and cost- and energy-efficient embedded processors has contributed to an increasing interest in adaptive stream mining (ASM) systems. In this class of signal processing systems, knowledge is extracted from data streams in real-time as the data arrives, rather than in a store-now, process later fashion. The evolution of machine learning methods in many application areas has contributed to demands for efficient and accurate information extraction from streams of data arriving at distributed, mobile, and heterogeneous processing nodes. To enhance accuracy, and meet the stringent constraints in which they must be deployed, it is important for ASM systems to be effective in adapting knowledge extraction approaches and processing configurations based on data characteristics and operational conditions. In this thesis, we address these challenges in design and implementation of ASM systems. We develop systematic methods and supporting

design tools for ASM systems that integrate (1) foundations of dataflow modeling for high level signal processing system design, and (2) the paradigm on Dynamic Data-Driven Application Systems (DDDAS). More specifically, the contributions of this thesis can be broadly categorized in to three major directions:

1. We develop a new design framework that systematically applies dataflow methodologies for high level signal processing system design, and adaptive stream mining based on dynamic topologies of classifiers. In particular, we introduce a new design environment, called the *lightweight dataflow for dynamic data driven application systems environment* (*LiD4E*). LiD4E provides formal semantics, rooted in dataflow principles, for design and implementation of a broad class of stream mining topologies. Using this novel application of dataflow methods, LiD4E facilitates the efficient and reliable mapping and adaptation of classifier topologies into implementations on embedded platforms.

2. We introduce new design methods for data-driven digital signal processing (DSP) systems that are targeted to resource- and energy-constrained embedded environments, such as unmanned areal vehicles (UAVs), mobile communication platforms, and wireless sensor networks. We develop a design and implementation framework for multi-mode, data driven embedded signal processing systems, where application modes with complementary trade-offs are selected, configured, executed, and switched dynamically, in a data-driven manner. We demonstrate the utility of our proposed new design methods on

an energy-constrained, multi-mode face detection application.

3. We introduce new methods for multiobjective, system-level optimization that have been incorporated into the LiD4E design tool described previously. More specifically, we develop new methods for integrated modeling and optimization of real-time stream mining constraints, multidimensional stream mining performance (e.g., precision and recall), and energy efficiency. Using a design methodology centered on data-driven control of and coordination between alternative dataflow subsystems for stream mining (classification modes), we develop systematic methods for exploring complex, multidimensional design spaces associated with dynamic stream mining systems, and deriving sets of Pareto-optimal system configurations that can be switched among based on data characteristics and operating constraints.

# DESIGN TOOLS FOR DYNAMIC, DATA-DRIVEN, STREAM MINING SYSTEMS

By

Kishan Palintha Sudusinghe

Dissertation 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
Doctor of Philosophy
2015

Advisory Committee:
Professor Shuvra S. Bhattacharyya, Chair/Advisor
Professor Steven Tretter
Professor Manoj Franklin
Professor Donald Yeung
Professor Rance Cleaveland

# Dedication

To my Dad and Mom

# Acknowledgments

I would like to give my sincere thanks to my advisor and mentor Prof. Shuvra Bhattacharyya for his invaluable guidance, support, encouragement, and inspiration. I am particularly grateful for believing in me and giving me plenty of opportunities and resources to be successful in my career as a PhD student. His unwavering support not only made this possible but also motivated me in many ways to be a disciplined and a productive researcher and a student. He has helped me in many ways and have motivated me during the difficult times in my life as a PhD student. Professor Bhattacharyya have assisted, guided, and prepared me for life beyond the PhD. Due to his attention to detail and very thorough and disciplined review process I have been able to publish quality publications in top-tier conferences with quality results. I also learned many important lessons from him that I have no doubt will be fruitful and pay dividends in future. For all this and many more, I am so grateful and thankful to you, Professor.

I am very much thankful to my PhD dissertation committee - Prof. Steven Tretter, Prof. Manoj Franklin, Prof. Donald Yeung, and Prof. Rance Cleaveland for being very flexible, accommodating, and above all for their reviewing of the thesis and valuable feedback.

The time I spent in DSPCAD research group under the leadership of Prof. Shuvra Bhattacharyya is one of the most memorable time in my life. I have formed many life-long connections and learned from many of the brightest and smartest people I have encountered in my life. Through many stages in my career as a

PhD student, current and past members of the research group have helped me enormously. I would like to thank Dr. William Plishker, Dr. Chung-Ching Shen, Dr. Nimish Sane, Dr. Hsiang-Huang Wu, Dr. Lai-huei Wang, and Dr. George Zaki for their advice and assistance during the early stages of my PhD. I thank Dr. Ilya Chukhman, Scott Kim, Yanzhou Liu, and Shouxin Lin for their support during the later stages in my PhD. I would also like to thank Yang Jiao, Lin Li, Kyunghun Lee, Haifa Ben Salem and Alexandre Mercat who have assisted me in many ways.

I would like to acknowledge Marshal Plan Scholarship foundation for awarding me with a scholarship to research abroad in Austria and my collaborators in University of Applied Sciences in Salzburg, Austria for welcoming me warmly and assisting me through out my stay in Salzburg as a PhD student and a visiting researcher.

I would also like to thank my brothers Shahan Sudusinghe and Emmash Sudusinghe, people closest to my heart, Anushka Imbuldeniya and Sachiko De Silva for believing in me and motivating me through out my life and especially during the PhD. They have been the pillars of support for me during the most difficult time in my life.

Finally and most importantly, I would like to thank my parents without whom my PhD would not have been a reality. They have tirelessly supported me and encouraged me to aim and reach higher. They have sacrificed many luxury and comfort in their life to give me this opportunity. I cannot thank them enough and this thesis and its dedication to them is a small thank you from me.

Last but not least, I thank God for giving me the opportunity and guiding me spiritually through out my PhD.

*The research underlying in this thesis was supported in part by the Air Force office of Scientific Research (AFOSR) and I would like to acknowledge their support*

# Table of Contents

# List of Figures

# Chapter 1: Introduction

## 1.1 Overview

With the increasing need for accurate mining and classification from a great variety of data content, and the growth of associated applications in mobile and distributed contexts, stream mining systems require increasing amounts of flexibility, extensibility, and adaptivity for effective deployment. In stream mining systems, knowledge is extracted from data streams in real-time as the data arrives, rather than in a store-now, process later fashion. Design and implementation of stream mining systems is complex due to various critical factors, including complex application requirements that involve constraints on energy efficiency, accuracy, and real-time performance; diverse operational platforms, such as resource-constrained sensor nodes, centralized and distributed servers, and hand-held computers (smartphones and tablets); and a wide variety of data content types that must be handled, including textual, speech, image, and video content, as well as data arriving from a growing variety of sensing devices.

At the same time, demand for real-time knowledge extraction from streams of data is increasing continually. This demand comes from a broad spectrum of application areas, including surveillance, cybersecurity, intelligent transportation, earth

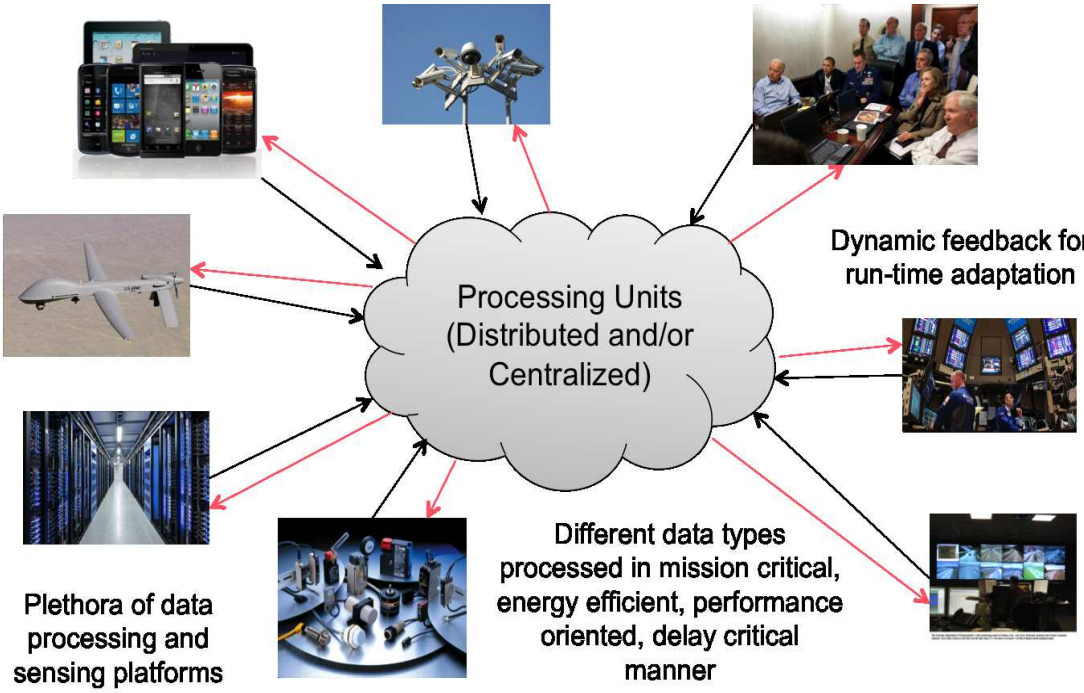


Figure 1.1: An illustration of the design space for adaptive stream mining systems.

science (e.g., water management and seismic monitoring), online financial analysis, manufacturing process control, multimedia search, health and life sciences, and medical services (e.g., see [1–3]). Figure 1.1 illustrates design challenges encountered and the complexity of the design space in the development of adaptive stream mining applications.

In this thesis, we address these challenges through an integration of methods in (1) adaptive classifier topologies for stream mining; (2) dataflow-based design and optimization of signal processing systems; and (3) dynamic data-driven applications systems (DDDAS).

Adaptive configuration of classifier topologies and operating points can be effective in enhancing performance trade-offs, such as trade-offs between classifier

accuracy and processing delay [2]. Here by classifier topologies, we mean networked interconnections of individual classifiers that cooperate on the given stream mining task, and that can be configured and connected in a modular fashion. In this context, one important aspect of modular configuration is the selection of specific operating points for the individual classifiers, which determine their associated trade-offs between detection probability and false alarm rate. A given classifier in general offers a range of such trade-offs, referred to as the detection error trade-off (DET) curve of the classifier.

In the context of signal processing applications, dataflow provides methods to represent, analyze, and optimize hardware/software systems in terms of graphical, data-driven models of computation [4]. In this modeling approach, application systems are represented as directed graphs in which graph vertices, called *actors*, represent computational components, and each edge represents the communication of data from an output of one actor to an input of another. A wide variety of techniques has been developed and continues to evolve for dataflow modeling, analysis of dataflow graphs, and mapping dataflow graphs into efficient implementations [5].

The DDDAS paradigm centers on the deep integration of instrumentation processes into application system design, and the use of such instrumentation to dynamically control how data is processed, and how future data is acquired, including how future rounds of instrumentation are configured [6, 7]. DDDAS methods are of increasing importance in part because ubiquitous sensors are now able to collect massive volumes of data for application systems, and tools are needed to efficiently and accurately determine which portions of this data are most relevant

to the current application context.

Previously, the areas of adaptive stream mining, dataflow, and DDDAS have evolved largely in independent directions. One important contribution of this thesis is to investigate synergies among these areas, and develop effective methods and tools for applying them in an integrated manner.

In the remainder of this chapter, we outline the specific contributions of this thesis.

## 1.2 Contributions

In this thesis, we have developed new techniques for design and implementation of adaptive stream mining application systems that are targeted to dynamic, resource-constrained, embedded environments. The tools and techniques developed in this work are not specific to any particular kind of stream mining application domain, and can be readily adapted across a wide range of areas that require real-time knowledge extraction from data streams. Our proposed new design tools are built to accommodate application-, algorithm-, instrumentation-, and design space-models, and integrate them in a systematic manner for optimized system design.

The contributions of this thesis are presented in three main parts — hierarchical dataflow modeling for dynamic, data-driven signal processing systems; design and scheduling for multi-mode stream mining; and multiobjective design optimization.

### 1.2.1 Hierarchical Dataflow Modeling

Prior work on signal processing oriented dataflow models of computation has emphasized static dataflow models, such as synchronous and cyclo-static dataflow [8–10]. In these models, the rates at which actors produce and consume data from their individual actors ports are deterministic, and known at compile time. Adaptive stream mining systems, however, require support for dynamic dataflow rates. Such support is needed, for example, to accommodate the flexibility required in handling multi-mode and dynamically reconfigurable stream mining behavior. Additionally, since our objective is to facilitate design of complex stream mining systems, support for hierarchical modeling is important for the model of computation that we employ. However, existing design methods based on dynamic dataflow graphs focus primarily on flat (non-hierarchical) representations (e.g., see [5]). With these motivations, we introduce in this thesis a novel, hierarchical, dynamic dataflow modeling approach called *hierarchical core functional dataflow* (*HCFDF*).

The HCFDF model of computation can be viewed as an integration of hierarchical semantics into the previously-developed *core functional dataflow* (*CFDF*) model of computation [11, 12], where the execution behavior of actors is determined dynamically through the use of distinct actor-level operational modes. More background on the CFDF model is described in Section 2.2.

Efficient integration of hierarchical semantics into CFDF involves a number of challenges that must be addressed, including the problem of ensuring that hierarchical embeddings of subsystems (subgraphs) adhere to the appropriate model

of computation within the subsystems in which they are embedded — that is, to ensure that when a HCFDF graph $G_1$ is nested within a higher level graph $G_2$, the interfaces of $G_2$ within $G_1$ conform to HCFDF semantics. Additionally, methods are needed to efficiently specify adaptive stream mining systems using the developed hierarchical modeling methods, and to map the resulting models systematically into optimized implementations.

To address these challenges, we have developed a novel design tool called *lightweight dataflow for dynamic data driven application systems* (*LiD4E*). LiD4E extends the lightweight dataflow environment (LIDE) [13, 14] with new capabilities for modeling, implementation, and design space exploration. These new capabilities are designed carefully to support the domain of adaptive stream mining systems, and more generally, for data-driven signal processing systems. The HCFDF model of computation, its realization in the LiD4E design tool, and the demonstration of LiD4E on practical stream mining systems form the first major contribution of this thesis.

The HCFDF model of computation and LiD4E design tool are presented in detail in Chapter 3.

### 1.2.2 Multi-mode Stream Mining

To exploit the potential of adaptive stream mining systems, designers must be able to efficiently model and integrate alternative classification *modes* for performing a given stream mining task (e.g., see [15]). Here, by a mode, we mean a distinct

classifier configuration or a distinct networked interconnection of multiple classifiers. For example, different modes may provide alternative trade-offs among metrics such as classification accuracy, false positive rate, energy efficiency, and real-time performance. The availability of alternative modes representing different trade-off points provides increased agility at runtime to automatically select a system configuration that is well matched to time-varying application conditions and constraints.

For example, when a surveillance system is in a state of "standby" operation (no potential threat has been detected), it is often desirable to operate in a mode that optimizes energy efficiency at some expense in classification accuracy and false positive profile. Once a potential threat has been detected, it is strategic to transition to a higher accuracy mode that has higher average power consumption. Furthermore, the degree to which energy efficiency should be traded off for accuracy during such a transition may depend strongly on constraints on available battery capacity (in case mobile sensor nodes are employed) or thermal considerations.

In the second major contribution of this thesis, we develop design methodologies and tools for scheduling and implementation of multi-mode adaptive stream mining systems on resource-constrained embedded processing platforms. Here, by *scheduling* we mean the assignment of computational tasks (modeled as dataflow graph actors) onto processing resources, and the ordering of actors that share the same processor. Scheduling is an important step in the mapping of applications onto embedded platforms as it affects many key implementation metrics, including latency, throughput, memory requirements, and energy consumption (e.g., see [16–20]). We introduce a scheduling framework called the *DDDAS-HCFDF-*

*multi-mode* (*DHMM*) framework, and demonstrate this framework by prototyping it within the LiD4E design environment. The DHMM framework provides a systematic approach for integrating instrumentation, classifier configuration, and run-time adaptation with efficient scheduling of embedded stream mining systems.

The DHMM framework is presented in detail in Chapter 4.

### 1.2.3 Multiobjective Design Optimization

Multiobjective optimization is important in the adaptive stream mining domain since applications in this domain must operate in the context of diverse operational constraints and optimization objectives. Metrics of interest include those related to real-time performance, including latency and throughput; those related to stream mining accuracy, such as precision and recall; and those related to energy efficiency, including peak and average power consumption.

In the third major contribution of this thesis, we develop new methods for multiobjective design optimization and design space exploration for adaptive stream mining systems. These methods build on the LiD4E design environment and DHMM scheduling framework described in Section 1.2.1 and Section 1.2.2, and are based on deeply integrated processes of instrumentation, measurement, profiling, and dynamic mode control processes to help designers navigate complex, multidimensional design evaluation spaces. Our new multiobjective design optimization approach, called *adaptive multiobjective design optimization* (*AMDO*), provides formal methods for constructing and manipulating parameterized stream mining systems, and

allows designers to efficiently explore and control the operational trade-offs that are realized by appropriate configuration of system-level adaptation methods.

The AMDO approach is is presented in detail in Chapter 5.

## 1.3 Outline of Thesis

The remainder of this dissertation is organized as follows. Chapter 2 provides background on various topics that are relevant for this research, including relevant background on dataflow models, the Dynamic Data Driven Applications Systems (DDDAS) paradigm, and machine learning. In Chapter 3, we present the hierarchical core functional dataflow (HCFDF) model of computation and lightweight dataflow for dynamic data driven application systems (LiD4E) design tool. In Chapter 4, we present the DDDAS-HCFDF-multi-mode (DHMM) scheduling framework. In Chapter 5, we present the adaptive multiobjective design optimization (AMDO) approach for multidimensional design space exploration and system-level optimization. We conclude in Chapter 6 with a summary of the developments in the thesis, and a discussion of directions for future work.

# Chapter 2: Background

## 2.1 Overview

In this chapter, we provide background on core concepts that are applied and built upon in the work presented in this thesis.

## 2.2 Core Functional Dataflow

Core functional dataflow (CFDF) is a dataflow model of computation that is geared toward design, analysis, and implementation of signal processing systems [11]. CFDF can be viewed as a programming model for developing signal processing components and systems that have statically known patterns of production and consumption rates (dataflow rates), as well as ones that employ dynamic dataflow rates.

In the context of signal processing system design, a dataflow graph consists a set of vertices and a set of edges, where the vertexes (*actors*) represent computational functions of arbitrary complexity, and *edges* represent first-in-first-out (FIFO) buffers that store data values as they pass between actors [4, 5]. Such data values are encapsulated in objects that are called *tokens*.

A dataflow edge can be represented as an ordered pair $e = (v_1, v_2)$, where $v_1$ and $v_2$ are actors in the enclosing dataflow graph. Here, $v_1$, denoted by $src(e)$, is called the *source actor* (or simply "source") of $e$, and $v_2$, denoted by $snk(e)$, is called the *sink actor* (or simply "sink") of $e$.

Execution of a dataflow actor is decomposed into a sequence of discrete units of execution, called *firings*, where each actor firing consumes and produces a well-defined number of (zero or more) tokens on each input and output port, respectively [4]. Different variants of dataflow have been developed for signal processing system design based on different restrictions on actor production and consumption rates. For example, in synchronous dataflow (SDF), production and consumption rates are fixed and statically known [8], and in cyclo-static dataflow (CSDF), production and consumption rates can vary dynamically but only according to periodic patterns of fixed, statically known rates [9].

A CFDF actor $A$ (dataflow graph functional component) has a set of $n \geq 1$ operational *modes* $S = \{m_1, m_2, \ldots, m_n\}$, and at any given time during execution of an enclosing dataflow graph, $A$ has a unique *current mode*. During each execution (dataflow firing), $A$ executes in its current mode, producing and consuming data on its dataflow ports, and updates its current mode. Each mode $m_i$ is restricted to have *synchronous dataflow* (*SDF*) semantics meaning that the number of data values produced or consumed from each actor port is constant [4]. However, different modes can have different production and consumption rates, which allows an actor to have dynamic (non-SDF) behavior across modes.

The specification of a CFDF actor includes two functions, *enable* and *invoke*.

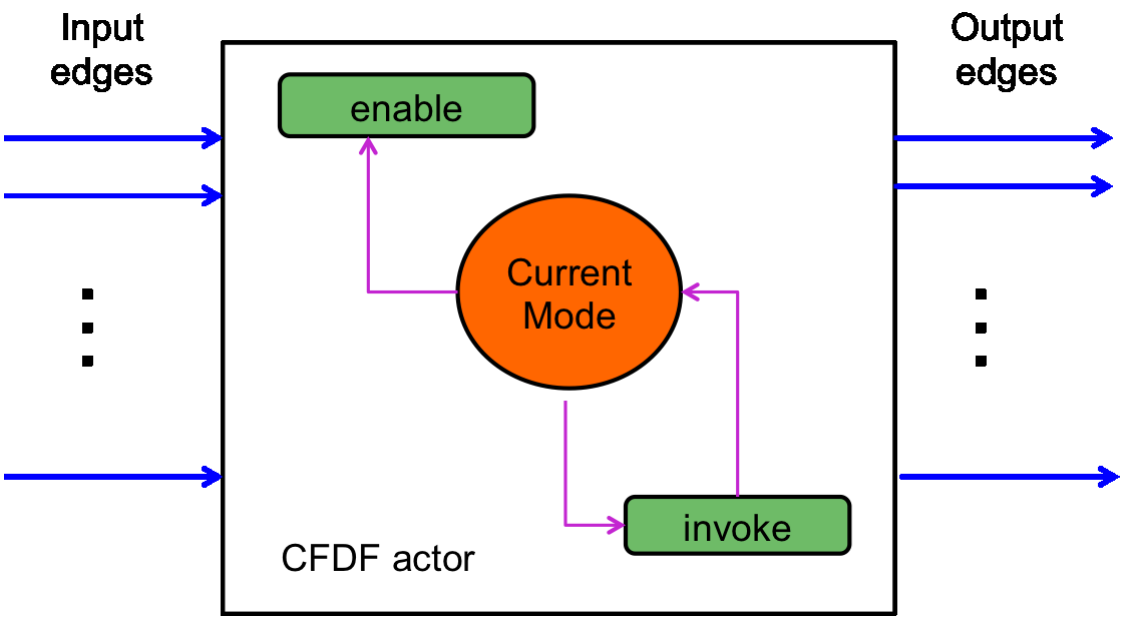


Figure 2.1: An illustration of a CFDF actor.

The Boolean-valued enable function returns `true` if the actor has sufficient data on its input ports and sufficient empty space on its output buffers to support a single firing in its current mode. The invoke function fires the actor in its current mode, and updates the current mode as determined as part of the firing [12].

Figure 2.1 shows an illustration of an actor that is designed using the CFDF model of computation. A CFDF actor can have any number of input ports and output ports as long as there is at least one input or output port. Each input or output port is connected to a dataflow edge in the enclosing dataflow graph where the actor is instantiated. As illustrated in Figure 2.1, the current actor mode is in general used by both the invoke and enable functions of the actor, and execution of the invoke function can change the current actor mode.

For a given CFDF graph $G$, a *schedule* is defined as a sequence of actor firings.

Given a set of buffer capacities (positive integer buffer sizes) for the edges, a schedule $S$ is *valid* if whenever an actor $A$ is fired in $S$, there is sufficient input data on all input ports of $A$ and sufficient empty space on all output ports of $A$ to support the mode of $A$ that is associated with the firing. In other words, if the enable function is executed just prior to any firing in a valid schedule, it should return `true`.

A periodic schedule for an SDF graph is a finite schedule that guarantees that every actor is fired at least once, the graph does not deadlock, and there is no net change in the number of tokens on any edge in the graph (i.e., the total number of tokens produced on each edge during the execution of the schedule is equal to the total number consumed from the edge). A periodic schedule can be iterated an arbitrary number of times with bounded memory requirements for the graph edges. This is an important property for signal processing applications, which must often operate on very large data streams, and on streams whose length (number of data samples) is not known in advance.

The existence of periodic schedules can be determined in finite time for SDF graphs and CSDF graphs, and when they exist for these classes of graphs, efficient algorithms are available for computing periodic schedules [8,10]. For many dynamic dataflow models of computation, including CFDF, the problem of whether or not dataflow graphs can be executed on unbounded streams with bounded memory requirements is undecidable [11,21,22]. This is a price designers pay for the added flexibility offered by such dynamic dataflow models.

## 2.3 Lightweight Dataflow Environment

The *lightweight dataflow environment* (*LIDE*) is a flexible design and implementation environment that allows designers to experiment with and explore dataflow-based techniques for design and implementation of signal processing systems [13,14]. LIDE is developed with a minimalistic approach that focuses on essential application programming interface (API) features for signal processing oriented, dataflow-based development. The framework provides capabilities for implementing signal processing systems in a variety of programming languages, and across a variety of platforms, including field programmable gate arrays, graphics processing units, desktop workstations, and programmable digital signal processors.

LIDE includes a number of libraries of dataflow graph element (actor and edge) implementations. The available library components include different kinds of FIFO implementations, basic signal processing actors, and actors for interfacing with input and output files. Using LIDE, designers can flexibly develop their own dataflow graph elements, other application-specific functional components (e.g., control-, parameterization-, and instrumentation-related modules), and schedulers to construct specialized dataflow-based systems.

For example, the work presented in this thesis includes prototypes of new design tools, dataflow graph elements, and schedules that are developed for efficient support of the ASM domain.

Design of actors within the LIDE framework is based on the semantics of the CFDF model of computation described in Section 2.2. LIDE also applies the

*lightweight dataflow* (LWDF) programming methodology, which provides a set of abstract application programming interfaces (APIs) that allow designers to construct, connect, and manipulate CFDF-based actor implementations [14]. LIDE provides concrete realizations of the LWDF programming methodology in a number of programming environments, including C, CUDA, and Verilog. The work presented in this thesis employs a C-language realization of the LWDF programming methodology, called LIDE-C, to implement and experiment with ASM systems.

When designing actors in LIDE-C, we use four interface functions, called the construct, enable, invoke and terminate functions. Implementation of an actor in LIDE-C centers on implementation of these interface functions for the actor. The construct function creates an instance of the actor and connects the ports of the actor to a set of edges that is passed through the function argument list. Using this mechanism of connecting actor ports to edges, the designer can build up the necessary topological connections between inputs and outputs of different actors in a graph. The construct function can also be utilized to perform any other pre-execution initialization tasks associated with a particular actor — i.e., tasks that are carried out before the first firing of the actor in an execution of the enclosing dataflow graph.

Conversely, the terminate function allows the designer to "close-out" aspects of the underlying actor, including deallocation of relevant storage objects, once the actor is no longer needed in the context of its enclosing graph. For example, the terminate function for an actor $A$ could be called once the graph has finished execution or when the graph is being reconfigured in such a way that $A$ is no longer

```
lide_c_gfilter_context_type *lide_c_gfilter_new(int tileX, int tileY,
        lide_c_fifo_pointer coef_in, lide_c_fifo_pointer tile_in,
        lide_c_fifo_pointer tile_out);

boolean lide_c_gfilter_enable(lide_c_gfilter_context_type *context);

void lide_c_gfilter_invoke(lide_c_gfilter_context_type *context);

void lide_c_gfilter_terminate(lide_c_gfilter_context_type *context);
```

Figure 2.2: An illustration of function prototypes for designing a CFDF actor.

used in the graph.

The enable and invoke functions for a LIDE-C actor provide implementations of the abstract CFDF enable and invoke semantics, respectively, that are described in Section 2.2. Intuitively, the enable function provides a lightweight mechanism to check at run-time whether or not a given actor is fireable — i.e., whether there is enough input data and empty buffer space to support the next firing of the actor. The invoke function provides the functionality to perform a single firing for the actor or a single block of firings if actor-level vectorization or "scalable dataflow semantics" are employed [23–25].

As an illustration of an actor interface in LIDE-C, Figure 2.2 shows prototypes of the construct, enable, invoke, and terminate functions, respectively, for a LIDE-C library actor that performs Gaussian filtering in an image. A more detailed example of a LIDE-C actor implementation will be discussed in Section 2.6.

## 2.4 Dynamic Data-Driven Applications Systems

The growth in technologies for sensing and computation has contributed to large increases in the volume of data that must be managed and analyzed in many kinds of applications. The deployment of such "big data" applications across distributed embedded systems results in the need to extract information efficiently from streams of data while adhering to complex challenges in networked, resource-limited, real-time environments. The paradigm of *dynamic data-driven applications systems* (*DDDAS*) is important in addressing such design challenges, which are central to the application domain of adaptive stream mining systems that is targeted in this thesis (e.g., see [26]).

DDDAS is a paradigm in which models are analyzed or simulated concurrently with executing applications, and measurements are dynamically monitored and adapted to help ensure that the models can be used effectively to control how the application is configured (e.g., what kinds of algorithms are employed, and the settings of the algorithm parameters), and how it executes on the target platform [7]. This leads to a deep integration of modeling, instrumentation, measurement, and measurement-driven adaptation throughout the design process. Furthermore, the notion of adaptation is extended to apply with equal emphasis to both the application configurations, and the instrumentation configurations (e.g., which sensors to monitor, what resolutions to employ, and how to filter the input data). In contrast, instrumentation is often largely ignored or treated as an afterthought or "side issue" in conventional design processes.

An important aspect of the DDDAS paradigm is the ability to integrate data dynamically into an application. Such data can be raw data that needs storage or further processing by the application (e.g., speech, image or video signals) or instrumentation data (e.g., battery capacity data, data from monitoring communication channel conditions, and information about resource utilization in the underlying processing platform). DDDAS also involves the ability to dynamically steer the control of an application system based on the results of specific subsystems, including subsystems that are dedicated to instrumentation. Such control can be carried out in an effort to best align the current application, platform and instrumentation configurations with the region of the overall application design space that the system is currently operating in. This design space region is estimated or predicted by the models that are managed concurrently with the executing applications as part of the DDDAS methodology.

Figure 2.3 shows an illustration of representative applications, algorithms, models, and design space components in the use of DDDAS methods for design and implementation of adaptive stream systems.

The DDDAS paradigm is relevant across a wide range of application areas. Examples include weather analysis [27], image guided neurosurgery [28], 3D tracking [29], and financial modeling. For further discussion on the use of DDDAS methods in various application domains, we refer the reader to [7].

The work presented in this thesis focuses specifically on the modeling and design space aspects illustrated in Figure 2.3. Through our emphasis on model-based design methodologies, specifically in the context of dataflow models of computation,

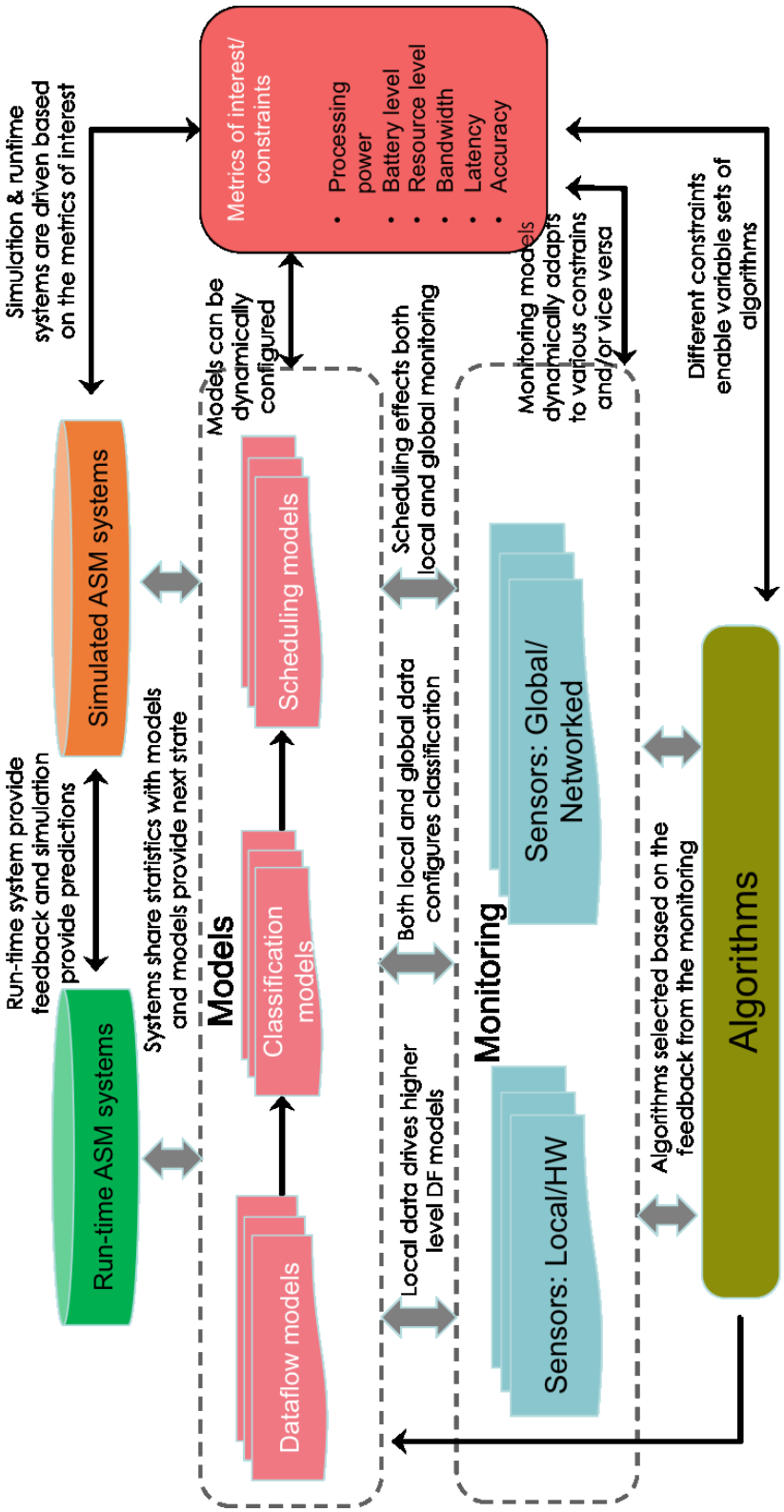


Figure 2.3: An illustration of the DDDAS paradigm applied to adaptive stream mining systems.

we develop novel modeling formulations and novel ways of applying models in the areas of application specification, instrumentation subsystem design, system-level adaptation, and task scheduling. These aspects of our contributions are emphasized in Chapter 3 and Chapter 4.

By the design space of an adaptive stream mining system, we mean the set of all system configurations — including all valid combinations of application-, algorithm-, and implementation-level configurations — that can be employed when the system is operating. Each element of the design space influences system performance in terms of the relevant design metrics, which in our demonstrations and experiments include metrics that relate to processing speed, energy consumption, real-time performance, and stream mining accuracy. The design evaluation space associated with a given design space and a given set of design metrics is the set of achievable performance levels in terms of all of the metrics. Thus, for example, if each design metric is measured as a real number and there are $n$ metrics, then the design evaluation space can be represented as a subset of $\mathbb{R}^n$, where $\mathbb{R}$ is the set of real numbers.

Adaptive stream mining systems and many other classes of applications systems are characterized by complex, multidimensional, design evaluation spaces. An important challenge is the development of design methodologies and tools that allow designers to understand and navigate these spaces and their associated design spaces so that they can select and deploy the most strategic operational trade-offs for the application scenarios of interest. For example, two common scenarios of interest in surveillance applications are (1) low-performance, low-fidelity standby states are that are designed to consume minimal energy and trigger an alert when a possible

threat is detected, and (2) high-performance, high-fidelity alert states that perform more intensive monitoring, analysis, and communication under situations when a threat is suspected.

In Chapter 5 of this thesis, we develop new methods to model and systematically navigate complex, multidimensional design evaluation spaces associated with adaptive stream mining systems.

## 2.5 Support Vector Machines

Support vector machines (SVMs) provide a supervised learning approach in machine learning that is used for classification and regression analysis. SVMs have gained significant popularity over recent years for their effectiveness, and amenability to mathematical analysis [30–32]. An SVM can be characterized as a linear algorithm in a high-dimensional feature space. Although SVMs are widely used for linear classification, they can also perform non-linear classification tasks efficiently through the use of a certain form of function called a *kernel function* [32]. The experimental adaptive stream mining systems developed in this thesis involve extensive use of non-linear classification through use of Gaussian radial basis functions (RBFs) as kernel functions.

Figure 2.4 illustrates how a kernel function maps instances from the SVM input space into a higher dimensional feature space, and how classification is carried in the feature space using hyperplanes that separate elements of different classes. For further background on SVMs, we refer the reader to [30, 32].

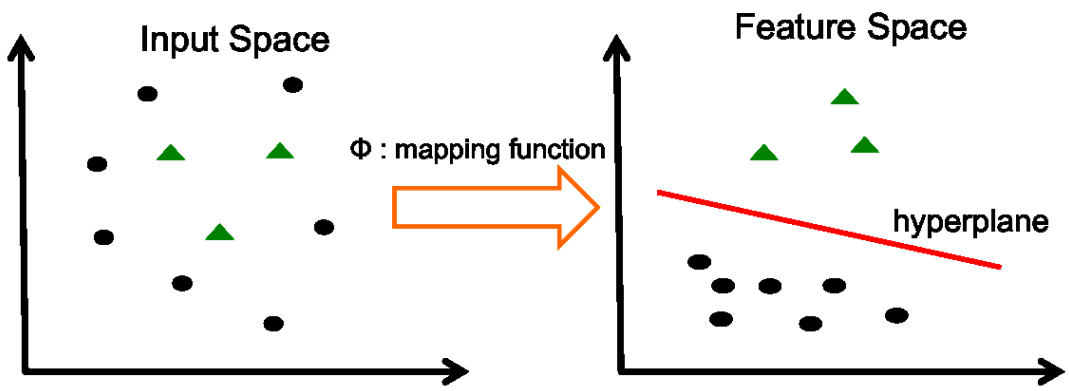


Figure 2.4: An illustration of an input space, feature space, and kernel function (mapping function) in SVMs.

## 2.6 SVM Implementation in LIDE

As described previously, the experimental adaptive stream mining systems developed in this thesis, including the SVM subsystems on which these systems are constructed, are programmed using LIDE-C, which is the integration of the lightweight dataflow programming methodology with the C programming language. In this section, we outline and illustrate our approach to implementing SVMs using LIDE-C.

We implement an SVM classifier in LIDE as an individual actor. Topologies of classifiers that provide complex functions using networked connections of SVMs can then be constructed by instantiating multiple SVM actors in LIDE-C and connecting them with lightweight dataflow edge implementations. Using LIDE as a foundation, we have developed additional domain-specific features for adaptive stream mining system implementation (beyond the development of SVM actor li-

brary components). The resulting design environment, called *lightweight dataflow for dynamic data driven application systems* (*LiD4E*) is presented in Chapter 3, and the contributions in Chapter 4 and Chapter 5 represent extensions to LiD4E to provide more advanced capabilities, including multi-mode stream mining system design, and multiobjective optimization.

Figure 2.5 shows an outline of the construct function for our SVM actor implementation. The first four function parameters, `input_data`, `input_svs`, `input_alphas`, and `output_class`, correspond to the FIFOs associated with the dataflow edges that are to be connected to actor ports in the enclosing dataflow graph. The inputs that are processed by the actor include the raw data that needs to be classified, and other SVM classifier parameters, such as the number of support vectors, alpha, bias, and sigma (from the RBF kernel). The classified class data is produced on the output FIFO labeled `output_class`.

Note that not all of the actor ports are accessed during all of the CFDF modes. For example, the SVM parameters are used only during a mode that is designated for configuring or "loading" the actor parameters. When the actor is invoked in its main data processing (stream mining) mode, it does not access the input ports that are used to load actor parameters. Also, note that the process of instantiating an actor includes initializing function pointers in the context to the addresses of the enable and invoke functions for the actor.

Figure 2.6 shows an outline of the enable function for our SVM actor implementation.

An outline of the invoke function for our SVM actor implementation is shown

in Figure 2.7. This illustrates the general convention in CFDF where the invoke function executes an unconditional actor firing (regardless of the state of the input and output FIFOs). It is the scheduler's responsibility to ensure that there is enough data and output space, which it can do efficiently by using the enable function as needed (or through static analysis if parts of the schedule or all of the schedule are being constructed statically). If the invoke function is called when there is insufficient data on the actor inputs or insufficient space on the outputs, then the results are unpredictable, and are likely to lead to incorrect application behavior. The potential for such problems needs to be addressed in the verification or testing processes for the actor. Efficient methods for validating implementations of CFDF actors are developed in [33].

Finally, the terminate function for our SVM actor implementation is shown in Figure 2.8. The terminate function is responsible for releasing memory that has been allocated during construction and execution of the actor. Typically, actor firings do not allocate memory, and the construct function is the source of the memory allocations that have to be considered in the terminate function. However, complex actors that maintain large and varying amounts of state may employ memory allocation (and even deallocation) within the actor firings (invoke functions). Such use of memory should be taken into account when implementing the terminate function.

## 2.7 Summary

In this chapter, we have reviewed various concepts and methods as background for the work that is presented in the subsequent chapters of this thesis. We have reviewed the core functional dataflow (CFDF) model of computation, and the lightweight dataflow environment (LIDE), which is a software package that supports design and implementation of signal processing systems in terms of CFDF models. We have also reviewed concepts of dynamic data-driven applications systems (DDDAS), which is a novel paradigm that promotes deep integration of models, instrumentation, measurements, and data-driven control of system configurations. Finally, we have reviewed support vector machines (SVMs), and illustrated the implementation of an SVM actor in LIDE.

```
lide_c_test_svm_context_type *lide_c_test_svm_new(
        lide_c_fifo_pointer input_data, lide_c_fifo_pointer input_svs,
        lide_c_fifo_pointer input_alphas, lide_c_fifo_pointer output_class,
        float bias, float rbf_sigma, int num_svs, int num_dims) {

    lide_c_test_svm_context_type *context = NULL;
    int i = 0;

    context = lide_c_util_malloc(sizeof(lide_c_test_svm_context_type));

    /* Initialize the CFDF mode of the actor*/
    context->mode = LIDE_C_TEST_SVM_MODE_LOAD;

    /* Setup the enable function for the actor */
    context->enable =(lide_c_actor_enable_function_type)lide_c_test_svm_enable;

    /* Setup the invoke function for the actor */
    context->invoke = (lide_c_actor_invoke_function_type)lide_c_test_svm_invoke;

    /* Setup actor parameters */
    context->num_dims = num_dims;
    context->num_sv = num_svs;
    context->bias = bias;
    context->rbf_sigma = rbf_sigma;

    /* Data specification */
    context->sv = lide_c_util_malloc(sizeof(float) * context->num_sv);
    …

    /* Files specification */
    context->input_svs = input_svs;
    …
    return context;
}
```

Figure 2.5: An outline of the construct function for an SVM actor.

```
boolean lide_c_test_svm_enable(lide_c_test_svm_context_type *context) {
    boolean result = FALSE;

    switch (context->mode) {
    case LIDE_C_TEST_SVM_MODE_LOAD:
        result = lide_c_fifo_population(context->input_data) >=
                    (context->num_dims) &&
                    lide_c_fifo_population(context->output_class) <
                    lide_c_fifo_capacity(context->output_class);
        ...
        break;
    case LIDE_C_TEST_SVM_MODE_PROCESS:
        result = 1;
        break;
    default:
        result = FALSE;
        break;
    }
    return result;
}
```

Figure 2.6: An outline of the enable function for the SVM actor example.

```
void lide_c_test_svm_invoke(lide_c_test_svm_context_type *context) {
    ...
    switch (context->mode) {
    case LIDE_C_TEST_SVM_MODE_LOAD:
        /* Read entries of testing data */
        <core computational task for data read>

        /* Switch Mode */
        context->mode = LIDE_C_TEST_SVM_MODE_PROCESS;
        break;
    case LIDE_C_TEST_SVM_MODE_PROCESS:
        /* Calculate the primary classification value */
        <core computational task for classification>

        /* Switch Mode */
        context->mode = LIDE_C_TEST_SVM_MODE_LOAD;
        break;
    default:
        context->mode = LIDE_C_TEST_SVM_MODE_ERROR;
        break;
    }
    return;
}
```

Figure 2.7: An outline of the invoke function for the SVM actor example.

```
void lide_c_test_svm_terminate(lide_c_test_svm_context_type *context) {
    free(context);
    free(sv);
    …
}
```

Figure 2.8: The terminate function for the SVM actor example.

# Chapter 3: Framework for Design and Implementation of ASM Systems

## 3.1 Introduction

As motivated in Chapter 1, Adaptive Stream Mining (ASM) is an important area of research in multimedia signal processing with the growing demand and deployment of multimedia and text based content in distributed processing nodes, and strict requirements for information being readily available in such nodes. Real-time knowledge extraction and classification is of high importance in a wide spectrum of multimedia processing application areas, including surveillance, cyber-security, transportation (e.g., intelligent traffic control), earth science (e.g., water management, seismic monitoring), online financial analysis, manufacturing process control, multimedia search engines, health and life sciences, and medical services [1–3].

As processing nodes become distributed and mobile, static, database oriented (e.g., query-driven) approaches to stream mining do not scale well. Additionally, distributed environments pose stringent constraints on system design for high volume multimedia content and energy efficient operation. Such systems need significant adaptivity to meet the dynamically changing needs of a wide range of users oper-

ating in a correspondingly wide range of environments, and employing devices that utilize diverse processing platforms.

In such distributed and dynamic operating environments, there is no notion of steady state processing or equilibrium that a system design can be targeted for. In particular, the operations that are most efficiently suited to the dynamically varying data characteristics must be configured carefully at run-time. Furthermore, there may be significant opportunity to streamline system operation by tuning the underlying embedded software structure (e.g., the task scheduling and memory management configurations) at run-time based on dynamically changing data characteristics, environmental conditions, and operational constraints. Novel design and implementation techniques are needed to address this growing need for adaptive, performance- and energy-optimized implementation of multimedia stream mining systems in the context of dynamic, data-driven processing scenarios.

In this chapter, we address this problem by introducing new design methods and an associated design tool that targets efficient implementation of data-driven, multimedia ASM systems, which is an emerging class of important applications in the broader area of *dynamic, data-driven application systems* (*DDDAS*) [7]. Our approach is based on new techniques involving signal processing oriented dataflow models of computation (e.g., see [5]). We introduce a novel dataflow modeling technique, *Hierarchical Core-Functional Dataflow* (*HCFDF*), to address the challenges mentioned above pertaining to ASM applications. Our technique is especially tailored for classification and knowledge extraction, and for helping designers to understand the parallel and distributed structure of ASM systems, and map these struc-

tures into efficient implementations. Based on this modeling technique, we introduce a unified design and implementation framework, called the *Lightweight Dataflow for Dynamic Data-Driven Application Systems Environment* (*LiD4E*), that can be used to systematically integrate adaptivity and data-driven system-level reconfiguration into arbitrary stream mining algorithms. We demonstrate key features of this tool through a face detection application.

Material in this chapter was published in preliminary form in [34].

## 3.2 Related Work

The design methodology presented in this chapter belongs to the broad class of dataflow-based design techniques for digital signal processing (DSP) systems (e.g., see [4, 5]). In such dataflow techniques, DSP systems are represented by designers in terms of dataflow graphs in which graph vertices (actors) represent signal processing tasks of arbitrary complexity, and edges represent logical communication channels between pairs of actors. Unlike conventional, compiler-oriented uses of dataflow, where it is employed as an intermediate representation, signal processing oriented dataflow techniques use dataflow as a programming model with semantics that are matched carefully to the specific DSP application domains to which they are targeted [5]. The work presented in this chapter provides the first systematic integration of such dataflow-based design and implementation with the emerging domain of multimedia adaptive stream mining systems. Further aspects of the novelty in our new dataflow based design methods are discussed in Section 3.3 and

Section 3.4.

Some current stream mining systems take a database-centric approach where relational operators are applied to streaming data. Since relational operators are well-understood, the system can statically determine the optimal operator order and placement over the available resource topology. Although some systems like Telegraph CQ [35] provide dynamic adaptation to available resources, they do not factor in application knowledge to achieve the best resource-to-accuracy trade-offs.

A limitation of existing work in stream mining systems is that the resulting processing delay has seldom been analyzed and, when it has been, it has always been analyzed in steady-state, at equilibrium, after all processing nodes are configured. However, the equilibrium can often be elusive due to the dynamic arrival and departure of query applications. Hence, the delay incurred when dynamically reconfiguring the different classifiers' operating points and organizing the classifier topology, out of equilibrium, is very important for real-time stream mining applications because it reduces the amount of time available to process incoming data. This reconfiguration delay out of equilibrium must be considered when designing solutions for real-time stream mining systems. Through its deeply integrated support for classifier parameterization, dynamic classifier and topological adaptation, and dataflow graph scheduling, the design framework presented in this chapter provides a foundation for analyzing, experimenting with, and optimizing the costs and benefits of adaptation and reconfiguration in ASM implementations.

## 3.3 Hierarchical Core Functional Dataflow

Unlike traditional stream processing, where the pipeline is deterministic, we focus on applications where the pipeline is data dependent and dynamically changing. In this chapter, we introduce hierarchical semantics for the core functional dataflow (CFDF) model of computation, which was discussed in Chapter 2, and we demonstrate the utility of this new form of modeling in the design and implementation of ASM systems. We refer to the model of computation that results from the integration of this hierarchical semantics with CFDF as *hierarchical core functional dataflow* (*HCFDF*). CFDF has been demonstrated in prior work to be useful for multimedia system design [36], and our integration of hierarchical semantics in CFDF helps to enhance the scalability and flexibility of CFDF-based design methods, and provides a critical enabler for our proposed ASM-targeted implementation techniques.

A major motivation for HCFDF is to encapsulate dynamic and adaptive topologies (subsystems) of real-time ASM applications so that they can be controlled flexibly and precisely by other HCFDF subsystems. HCFDF allows for the unique dynamic dataflow modeling and scheduling features of CFDF to be applied hierarchically so that complex, data-driven ASM topologies can be composed and scheduled. The CFDF model has similarities to other dataflow models, such as *scenario-aware dataflow* (*SADF*) [37], and we envision that the framework developed in this chapter can be adapted to such related models. Such generalization is a useful direction for further work.

HCFDF generalizes CFDF, which is developed in [12] and introduced in section 2.2. HCFDF graphs extend CFDF semantics with a specialized class of *hierarchical actors*. A hierarchical actor $H$ in an HCFDF graph can be viewed as a CFDF actor that has an associated set of nested HCFDF subgraphs $\sigma = N_1, N_2, \ldots, N_c$, where $c$ is referred to as the *cardinality* of $H$. For each subgraph $N_i$, the specification of $H$ places the interface ports of $N_i$ in one-to-one correspondence with a subset of the actor ports of $H$. Furthermore, each operational mode $\mu_i$ of $H$ can invoke an arbitrary subset of the subgraphs within $\sigma$ with each subgraph being invoked any number of times. The key restriction here is that the net data production and consumption at the actor ports for each $\mu_i$ must be constant, so that the actor "appears from the outside" as a CFDF actor. Such constant net production and consumption can be guaranteed if the nested subgraphs are all SDF graphs and are all invoked constant numbers of times within each mode. However, more flexible forms of dataflow and data-dependent subsystem invocation rates can be incorporated as long as the net production and consumption volumes at the ports of $H$ are constant over the entire mode execution.

Figure 3.1 illustrates a pseudocode sketch of how subgraphs can be invoked flexibly within the specification of a hierarchical actor in HCFDF. In this example, we assume that `graph1` and `graph2` are both SDF graphs that have the same net production and consumption rates at their input and output interfaces. The data availability for each subgraph invocation is ensured as part of the CFDF enable function constraint for the given actor mode ($M_0$ in this case). The symbols $T_1$ and $T_2$ represent *subgraph termination conditions* (*STCs*), which are objects that

specify "how long" to execute each graph. For example, in a particular hierarchical instantiation, an SDF subgraph may have its STC specified as a positive integer *blocking factor*, which gives the number of iterations to execute for any minimal periodic schedule for the graph. Such a specification is unambiguous in terms of the well-defined concept of minimal periodic schedules in the SDF model of computation. The actual schedule that is used to execute each subgraph is *not* part of the schedule specification. The schedule can be computed or otherwise determined by the synthesis tool, at run-time or using a combination of synthesis-time and run-time techniques. Such separation of concerns between scheduling and system specification is a fundamental objective in dataflow-based signal processing environments (e.g., see [5]), which our HCFDF framework supports through its subgraph invocation interface.

The HCFDF model promotes modularity as specialized kinds of SDF graphs, as well other kinds of CFDF graphs, can be reused readily across different applications and across different subsystems of the same application by integrating them through hierarchical CFDF actors.

The formulation of HCFDF introduced in this chapter is inspired by the modeling techniques of *interfaced-based SDF*, and *heterochronous SDF* [38, 39]; however, HCFDF is unique in its provision for encapsulating and configuring arbitrary collections of SDF graphs, as well as more general forms of dataflow graphs, within individual hierarchical actors.

```
switch(mode) {
case M0:
    x = get_token(input1)
    if (x == TRUE) {
        subgraph_invoke(graph1, T1);
        next_mode = M1
    } else {
        subgraph_invoke(graph2, T2);
        next_mode = M2
    }
    break;
case M1:
    ...
    break;
case M2:
    ...
    break;
default:
    exception("Invalid actor mode");
}
```

Figure 3.1: Control of subgraphs in hierarchical CFDF actors.

## 3.4 Design and Implementation Framework

In this section, we present a new design tool called the *Lightweight Dataflow for Dynamic Data-Driven Applications Systems Environment* (*LiD4E*). LiD4E is developed as a design tool plug-in oriented toward ASM system design. Specifically, LiD4E is a plug-in to the previously developed *lightweight dataflow* (*LiDE*) framework [40], which provides features for design and implementation of signal processing systems using dataflow graphs (see Section 2.3). LiD4E incorporates a

first version implementation of HCFDF semantics to support the novel capabilities for ASM system design that are presented in this chapter.

To support the adaptive, dynamic, and autonomous nature of ASM applications, we include a set of new decision and control actors (dataflow graph functional components) in the LiD4E framework. These are called *adaptation manager*, and *adaptive classification module* actors. LiD4E also applies *instrumentation actors*, which were introduced in [36] as a means for integrating static and run-time instrumentation capabilities systematically within a formal dataflow context. In LiD4E, instrumentation actors are used to provide monitoring and feedback to drive the adaptation tasks in an overall ASM system. Because instrumentation actors are themselves genuine dataflow components, their functionality can be parameterized, adapted, scheduled, and transformed in a way that is full integrated with the overall system design. This allows the instrumentation mechanisms to adapt seamlessly and efficiently as needed by the deployed ASM system.

Figure 3.2 illustrates how adaptation mangers, adaptive classification modules, and instrumentation actors interact in the LiD4E framework. The bottom part of Figure 3.2 illustrates a stream of facial images being injected as input into the system, as we employ face detection as a demonstration application in Section 3.5. However, the overall ASM system design and implementation framework illustrated in Figure 3.2 can be applied to arbitrary media types for a wide variety of different classification tasks.

The blocks labeled Adaptation Manager, Instrumentation Actor, Performance Constraints, Feedback, and Output around the periphery in Figure 3.2 represent

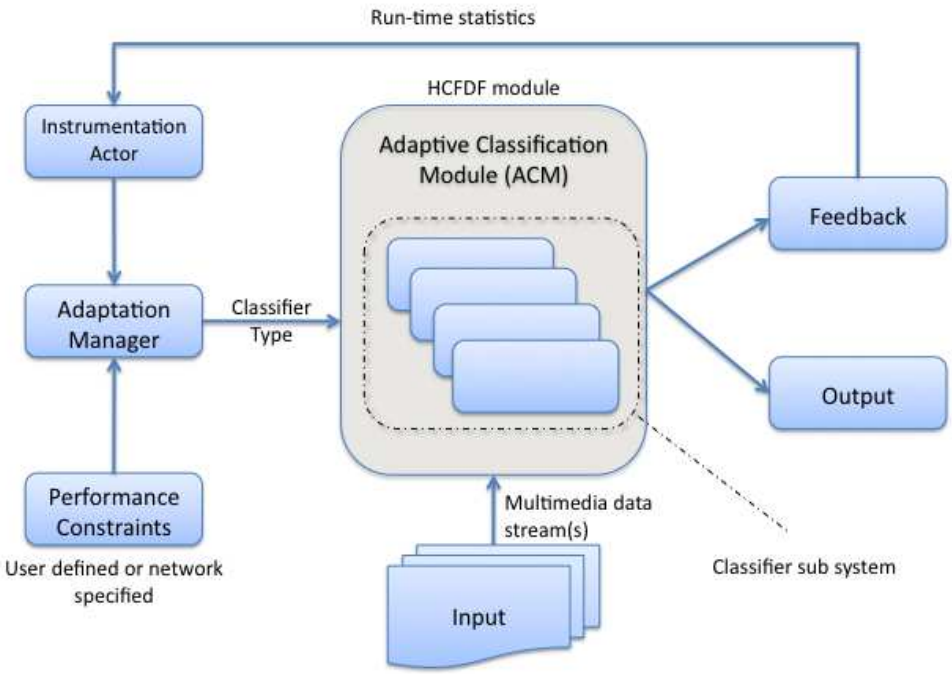


Figure 3.2: Overview of ASM system design in LiD4E.

standard components in our LiD4E-based ASM system design framework. While these actors can be instantiated and connected in arbitrary ways with other actors, the schema shown in Figure 3.2 can be viewed as a specific design pattern that can be used in LiD4E to implement a broad class of multimedia ASM systems.

To enable formal analysis, and systematic simulation and code generation in terms of the CFDF model of computation, these actors must adhere to CFDF semantics at their interfaces (see Section 3.3). Subject to this interfacing restriction, the actors can be developed using arbitrary platform-oriented languages (e.g., C, CUDA, Verilog or VHDL), and can employ arbitrary specialized algorithms and

implementation techniques within them to achieve the associated specialized parts of the overall ASM system functionality illustrated in Figure 3.2.

Intuitively, the Adaptation Manager provides front-end decision-making to select an appropriate classifier type based on the current performance constraints and feedback from instrumenting system performance and output values. As with the other blocks in Figure 3.2, this block can be implemented as a primitive CFDF actor or as a hierarchical actor that encapsulates an arbitrary collection of HCFDF subsystems.

The Instrumentation Actor collects runtime statistics of the system so that they can be monitored to drive system adaptation. For further details on the use of instrumentation actors in CFDF graphs, we refer the reader to [36].

The Performance Constraints actor provides a flexible CFDF interface to control the criteria under which the overall system will be adapted and optimized at run-time. These constraints can (1) be specified by a human through an appropriate user interface, (2) arrive through a communication network, (3) result autonomously from monitoring of system resources (e.g., battery levels), or (4) be formulated using any combination of these three general methods. In the version of the LiD4E framework that is described in this chapter, we support only the first method — user interface based constraint specification. Integration of a more rich class of constraint specification features is covered in the later chapters of this thesis.

The block in Figure 3.2 labeled Adaptive Classification Module (ACM) represents a hierarchical actor, which encapsulates a set $C$ of classifiers that are to be deployed as the "computational core" of the targeted ASM system. During a given

invocation of the ACM actor, an arbitrary subset of classifiers within $C$ can be selected, configured, connected, and operated in a cooperative manner (e.g., through the class of classification topologies studied in [2]). We elaborate more on design and application of the ACM actor in Section 3.5.

## 3.5 Case Study: Face Detection

In this section, we demonstrate the utility of the LiD4E environment and the underlying HCFDF model of computation with a case study centering on an ASM system for face detection.

### 3.5.1 Adaptive Classification

Adaptive classification systems are in general comprised of multiple classifiers with different parameters or performance goals (e.g., see [2]). In this context, we define "adaptive" as the ability of a system to change its operational parameters based on feedback or external input to maximize the effectiveness of the classification system, and select strategic combinations of classifiers dynamically through systematic processes to account for input data, operational constraints, and classifier characteristics. We define a "non-adaptive system" as one that operates using only one (static) set of parameters.

Support vector machines (SVMs) are supervised learning models that can be used for classification purposes (e.g., see [41]). A trained SVM classification model takes input data and calculates 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 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 (SVs). Nonlinear classification using SVMs can be effective for the task of face detection [42]. One of the most popular kernels in this context is the Gaussian radial basis function, defined by:

$$k(x_i, x_{SV}) = exp(-\gamma \|x_i - x_{SV}\|^2), \tag{3.1}$$

where $x$ represents the data point and $\gamma$ is a parameter that can be configured. By using the kernel trick (i.e., the method for mapping observations to an inner product space), an SVM model can be trained efficiently on the data. Cross-validation can be used to determine the optimal parameter values for each SVM based on the needs of the application.

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 HCFDF-based modeling of multi-modal SVM classification using the three classifiers in conjunction with the framework of Figure 3.2, and dynamic selection of the classifier to use based on situational goals. We design, implement, and experiment with this ASM system using the LiD4E environment.

In LiD4E, each subsystem can be operated as an independent dataflow graph with its own schedule or set of alternative schedules. This capability, along with the

associated classifier configurations, allows each subsystem to be specialized in terms of different operational qualities, such as high accuracy, low runtime, and low false positive rate. The focus at the subsystem level is to provide the desired functionality and associated operational qualities so that the subsystem can be selected when those qualities are determined to match best with the dynamic, data-driven needs of the application.

In this case study, each subsystem consumes input data — in the form of image pixels — and produces output data using an SVM classifier that is specialized (through its parameterization) for certain operational qualities, as described earlier. The common objective of all of the classifiers employed is to determine whether or not a given input image corresponds to a human face. The actors that are instantiated within the subsystems are the same; however, their parameters are configured in different ways to provide the desired trade-offs among operational qualities. Each subsystem consists of one actor to read the parameters of the associated classifier from a file, and another actor to perform classification using the SVM classifier. For embedded implementation when the parameter sets are of manageable size, the sets can be retrieved from memory rather than a file. Exploration in LiD4E with such resource-constrained, embedded implementation techniques is a useful direction for future work.

In our experiments, we provide streams of dynamically-varying application constraints, as explained before, as input to the ASM system. Such a "constraint stream" simulates the effect of dynamic, data-driven changes in operational requirements. Based on the most recently applied constraints to the system, one of the

classifiers $c_t$ is chosen whose associated operational features are best matched to the constraints. Subsequent input data is then directed to $c_t$ (and diverted from the other classifiers) until changes in constraints cause selection of a new classifier. We use a simple task scheduling strategy, called the *canonical scheduler*, for each classifier subsystem [12]. The different subsystem-level schedules are then integrated based on HCFDF semantics, as discussed in Section 3.3. A canonical schedule is easy to construct and understand, and is therefore useful during simulation and functional validation. Exploration of more sophisticated scheduling techniques in LiD4E, including techniques that are optimized for specific platforms, is a useful direction for future work.

### 3.5.2 Experimental Setup and Results

Our LiD4E-based ASM system was implemented for face detection on a 3GHz desktop computer with two Intel Xeon CPUs and 3GB RAM, and with Ubuntu 10.04 as the operating system. We used C as the language for programming individual actors, and used associated C-based libraries in LiD4E to provide HCFDF-based actor integration, dataflow buffer management, and scheduling. We used the `gcc version 3.4.4` compiler for the back end of the implementation process. We developed the SVM classifiers using MATLAB, trained the classifiers using the MIT CBCL face database [43], and then ported them to C for LiD4E-based actor implementation. The functional accuracy of LiD4E was verified through individual simulation of each of the SVMs using MATLAB. Each subsystem consisting of the

different SVM classifiers was checked to ensure that the input data only went to one of the subsystems. Test images from the MIT CBCL face database were used in the verification.

The overall functionality of our LiD4E-based ASM system implementation was validated by comparing if the correct sequence of classifiers was selected based on the applied sequence of application constraints, and comparing the results of the individual classifier invocations from LiD4E with the results of the corresponding MATLAB-based classifier prototypes. Table 3.1 shows the runtimes for several alternative configurations of our LiD4E-based ASM system. For each "single run" experiment, one 19x19 image was input into each classification subsystem with a specific operational constraint. The "stream run" data was collected using, for each experiment, a sequence of three 19x19 images. The row in Table 3.1 labeled "all modes enabled" gives aggregate results for executing all three classifiers on each of the three images in the input stream, while the other "stream run" rows correspond to application of individual classifiers in isolation.

With our ASM design framework, implemented systems can switch dynamically among modes so that the classifiers employed are best matched to the constraints and data characteristics presented by the external input. However, because different classifiers can be utilized, there may be a performance loss when using classifiers that have longer run-times than the fastest classifier available within the system. Such a trade-off may be desirable when there is a greater need for accuracy or a lower false positive rate. To quantify this trade-off, we compare each ASM system developed in LiD4E with a corresponding non-adaptive system that uses only

the fastest available classifier. The run-time aspect of this comparison is captured by the performance loss ratio (PLR), which gives the ratio of run-times between the classifier employed (numerator) and the fastest available one (denominator). The PLR quantifies the performance loss (in exchange for higher quality classification results) compared to a non-adaptive system that operates using only the fastest available classifier.

An average of 26% improvement in execution time per classifier invocation is observed in the stream version of the experiments. This improvement can be attributed to amortization of start-up and termination overhead of the LiD4E system, including program startup overhead associated with the underlying operating system, across longer input data sets. The overall benefits of such amortization increase with the length of the input stream. The overhead attributed to the LiD4E-based ASM framework (Figure 3.2) was measured as $(\tau - \kappa)/\tau$, where $\tau$ represents the total execution time of all of the actors in Figure 3.2, and $\kappa$ represents the execution time of the core classification tasks (the classifier actors). This overhead was measured to consistently fall in the range of 0.75%–1.1%, depending on the type of classifier employed. The overhead of our proposed ASM system design framework is thus found to be relatively small, and much smaller than the overall system overhead, which is inclusive of operating system overheads, as described above.

Most importantly, the results in Table 3.1 provide insight into the capability of our novel framework to dynamically adapt to trade-offs in performance and accuracy. They also demonstrate the ability to achieve improved performance through a unified design framework that adheres to relevant operational constraints.

| Execution Mode | Classifier Mode (seconds) | Average Execution Time | Standard Deviation | Performance Loss Ratio |
|---|---|---|---|---|
| Single Run | Mode with highest accuracy classifier | 0.492 | 0.080 | 1.469 |
| | Mode with fastest classifier | 0.335 | 0.064 | — |
| | Mode with lowest false positive classifier | 1.762 | 0.262 | 5.260 |
| Stream Run | All modes enabled | 2.116 | 0.297 | 3.182 |
| | Mode with highest accuracy classifier | 1.038 | 0.160 | 1.561 |
| | Mode with fastest Classifier | 0.665 | 0.104 | — |
| | Mode with lowest false positive classifier | 4.514 | 0.360 | 6.788 |

Table 3.1: Face detection run-times using LiD4E. The unit of run-time measurement here is *seconds*.

## 3.6 Summary

In this chapter, we have introduced LiD4E (lightweight dataflow for dynamic data driven applications systems environment) as a formal dataflow based design framework for multimedia adaptive stream mining (ASM) applications. Through an ASM application for face detection, we have demonstrated the capability of LiD4E to enable systematic trade-off exploration among ASM figures of merit, including classification accuracy, processing speed, and minimization of false positive rates. Furthermore, through their formal connection to dataflow foundations, ASM system models in LiD4E expose application concurrency while ensuring determinacy in the execution results regardless of which scheduling technique is employed. Through such scheduling flexibility, designers of ASM systems have the potential to further enhance trade-off exploration and run-time adaptation. Thus, building on the design methods developed in this chapter to explore new scheduling techniques for ASM systems is a useful direction for further investigation.

# Chapter 4: Model Based Design Environment for Data-Driven Embedded Signal Processing Systems

## 4.1 Introduction

Embedded systems are often deployed and configured to handle multiple application tasks concurrently across different subsets of processing resources. In the domain of embedded signal processing, modern platforms consist of multiple processing cores that can concurrently support DSP- (digital signal processing) intensive functions such as multimedia (e.g., face recognition, speaker identification, pattern recognition) and wireless communication (e.g., GSM, digital radio, NFC, Bluetooth), as well as control-oriented functions, such as those associated with user interfaces and file management (e.g., see [5]). With the increasing need for efficient and robust development of embedded systems, it is important to utilize and effectively manage the limited resources available in these computing devices dynamically in the context of data characteristics and operating conditions. Static modeling and management of execution constraints, including those involving energy consumption, real-time performance, computational resources, and core application accuracy, is not effective in designing efficient embedded systems that must adapt to time-varying require-

ments. Thus, in this chapter, we develop new techniques for dynamic, data-driven modeling, scheduling, monitoring, and execution of DSP applications running on resource limited embedded platforms.

Dataflow modeling techniques are widely used to model, schedule and implement DSP systems [5]. Adaptive Stream Mining (ASM) is an important subclass of DSP applications where real-time knowledge extraction and classification are of high importance [15]. Unlike traditional data mining systems, where data is stored statically and mined through queries on the static (or slowly changing) data, ASM data arrives continuously and must be processed in real-time. Statically configured approaches to ASM processing do not scale well, with scalability problems getting worse as ASM nodes become distributed and mobile. Furthermore, integrating diverse application subsystems or diverse configurations of the same subsystem (multi-mode operation) for trade-off optimization or information integration requires adhering to global constraints on resource utilization and performance, while managing different quality-of-service (QoS)characteristics of the subsystems. Therefore, novel design and implementation techniques that deviate from traditional, statically-oriented stream mining system design are needed to address the growing need for performance- and energy-optimized implementation of ASM in the context of dynamic, data-driven, and multi-mode processing scenarios. A conceptual design flow for this class of targeted multi-mode scenarios is illustrated in Figure 4.1.

This work represents a novel integration of dataflow based design methods for signal processing with the paradigm of Dynamic Data-Driven Application Systems (DDDAS). High-level, signal-processing-oriented dataflow models of computation al-

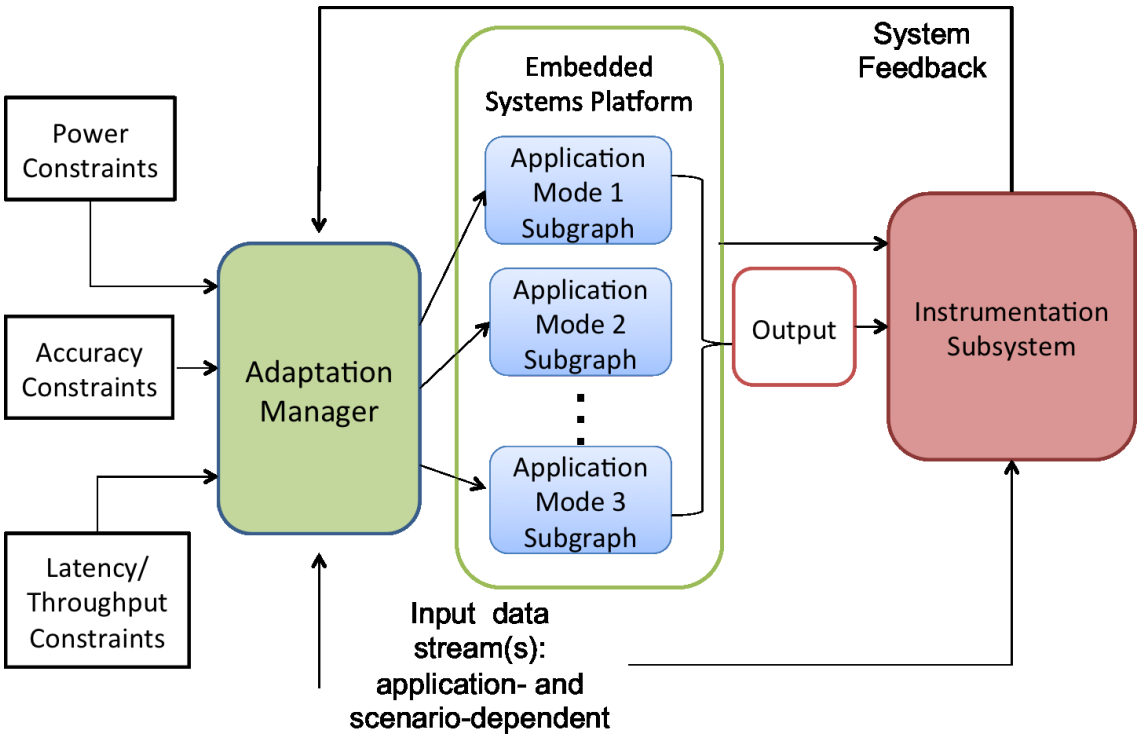


Figure 4.1: Data-driven, multi-mode embedded system design flow.

low designers to systematically formulate the design flow for a DSP system, and to integrate hardware, software, and application constraints into such design flows [5]. DDDAS is a paradigm that rigorously integrates application system modeling, instrumentation, and dynamic, feedback-driven adaptation of model and instrumentation parameters based on measured data characteristics [7]. In this work, we combine the methodology of dataflow-based DSP system design with the DDDAS paradigm to address the novel constraints and challenges of real-time, multi-mode ASM processing on embedded platforms. Our proposed new design framework provides a structured approach for design, implementation and optimization of ASM systems under stringent platform constraints and dynamically-changing application

requirements and data characteristics.

To address the design and implementation of multi-mode ASM systems, we apply in this chapter our previously introduced (see Section 3.3) dataflow modeling technique called *Hierarchical Core Functional Dataflow* (*HCFDF*) [34]. In particular, we present a novel application of HCFDF to efficiently model and manage multi-mode application scenarios. In this modeling approach, dynamic adaptation is represented through hierarchical inclusion of a special kind of actor (dataflow-based software component) called a *decision actor*. Such hierarchical use of decision actors is employed to switch among application subsystems based on data-driven demands involving performance-energy tradeoffs.

We also apply the HCFDF model to develop new methods for performance-energy-aware, dynamic scheduling of application subsystems. These scheduling techniques are geared towards efficient, context-aware adaptation of embedded DSP systems in multi-mode design scenarios. We integrate our new scheduling techniques with DDDAS concepts to introduce a unique model-based design environment for data-driven resource, constrained DSP applications. This design environment is prototyped and demonstrated by building on the *Lightweight Dataflow for Dynamic Data-Driven Application Systems Environment* (LiD4E), which is a tool for experimentation with and optimization of dataflow-based design methods for ASM systems [34] that is described in detail in Chapter 3

Material in this chapter was published in preliminary form in [44].

## 4.2 Related Work

As mentioned in Section 4.1, the work presented in this chapter is rooted in core concepts of the DDDAS paradigm [7], and of dataflow-based design for DSP systems (e.g., see [4, 5]). In DSP-oriented dataflow modeling, applications are represented in terms of *dataflow graphs*, where graph vertices (*actors*) represent signal processing tasks of arbitrary complexity, and edges represent logical FIFO communication channels between pairs of actors. In this work, we apply dataflow as a programming model with semantics that are carefully matched to the targeted DSP application domain — i.e., dynamic, data-driven signal processing systems [5, 34], and more specifically, adaptive stream mining systems. This modeling approach differs from uses of dataflow as a compiler intermediate representation (e.g., see [45]), and as a form of computer architecture [46].

The work presented in this chapter builds upon our previous work on adaptive stream mining systems for multimedia applications [34]. This chapter goes beyond the previous work described in chapter 3 by investigating design and implementation problems for multi-mode applications, and by developing new scheduling techniques for mapping applications onto embedded platforms while monitoring and managing dynamically-changing data characteristics and operational constraints.

Various studies on embedded stream mining have focused on performance optimizations for specific applications (e.g., see [47–49]). Similarly, the works of [48,49] provide generalized scheduling and design strategies respectively, but focus on statically configured systems, without emphasis on handling time-varying data charac-

teristics. Work in this chapter is distinguished from these prior efforts in our focus on multi-mode application systems, and the integrated application of dataflow and DDDAS principles to such a multi-mode context.

Another relevant direction of prior work has involved the incorporation of data-driven adaptability to individual signal processing functional components (dataflow actors and their underlying algorithm parameters). For example, the works presented in [50–52] have studied such capabilities for speech processing applications. Here, adaptability is achieved by dynamically updating the key signal flow graph components, such as Hidden Markov Models (HMMs), linear predictive coding (LPC) blocks, and Mel-Frequency cepstral coefficients (MFCC) within a given speech recognition application [50, 51]. These methods are relevant to diverse applications, including speaker verification, audiovisual forgery, and low bit rate speech coding. The methods can provide useful building blocks (parameterized actor and subsystem designs) for the directions that we pursue in this chapter. However, the approach that we pursue in this chapter is more flexible in terms of data-driven operation since we consider adaptation of application models globally (at the dataflow graph and scheduling level) as well as locally (at the level of individual actors or subsystems).

## 4.3 Application Modeling

In this section, we discuss the modeling methods applied in our new design environment for data-driven signal processing systems, and we demonstrate how

they can be applied to the design of multi-mode application systems. These modeling methods are supported by the the LiD4E design tool, which is introduced in Section 4.1, and provides a foundation for our prototyping of and experimentation with the methods developed in this chapter. While the underlying modeling foundation (HCFDF semantics) reviewed in this section has been developed in chapter 3, our application of HCFDF semantics to multi-mode applications is a novel aspect described in this chapter.

A key feature of LiD4E is the provision for signal processing pipelines (i.e., chains of signal processing modules, such as classifiers, digital filters and transform operators) that can be data dependent and dynamically changing. LiD4E employs hierarchical core functional dataflow (HCFDF) semantics as the specific form of dynamic dataflow [34]. Through its emphasis on supporting structured, application-level dynamic dataflow modeling, HCFDF provides a formal, model-based framework through which applications in DSP and related domains can be designed and analyzed precisely in terms of integrated principles of DDDAS and dataflow.

In HCFDF graphs, actors are specified in terms of sets of processing modes, where each mode has static (*dataflow rates*) — i.e., each mode produces and consumes a fixed number of data values (tokens) on each actor port. However, different modes of the same actor can have different dataflow rates, and the actor mode can change from one actor execution (*firing*) to the next, there by allowing for dynamic dataflow behavior (dynamic rates). Additionally, HCFDF allows dataflow graphs to be hierarchically embedded (nested) within actors of higher level HCFDF

graphs, thereby allowing complex systems to be constructed and analyzed in a scalable manner. The design rules prescribed for hierarchical composition in HCFDF graphs ensure that actors at each level in a design hierarchy conform to the semantics of HCFDF or some restricted subset of HCFDF semantics, such as cyclo-static dataflow (CSDF) or synchronous dataflow (SDF) [8, 53]. For further details on HCFDF semantics, we refer the reader to [34].

As demonstrated preliminary in [34] and described in chapter 3 of this thesis, HCFDF modeling enables run-time adaptation of signal processing topologies, including dataflow graphs that are constructed using arbitrary combinations of classifiers, filters, and transform units. Through the inclusion of a special HCFDF design component called an *adaptive classification module* (*ACM*), the designer can invoke multiple operating modes at run-time, and selection of such operating modes can be driven based on system feedback — e.g., based on instrumentation that monitors data characteristics, and guides selection based on desired trade-offs among performance, accuracy, and energy consumption.

To apply such a hierarchical, DDDAS-based dataflow design methodology to the multi-mode application scenarios targeted in this chapter, we represent a system design as a set of mutually exclusive application modes $S_M = \{\mu_1, \mu_2, \ldots, \mu_N\}$, where each $\mu_i$ represents a set of application subsystems that are active during the corresponding mode together with the configurations (actor-, application- and schedule-level parameters) that are to be applied to the subsystems whenever $\mu_i$ executes. This is illustrated in Figure 4.2. Although execution across the $\mu_i$s is carried out sequentially, based on an ordering that can be determined dynamically,

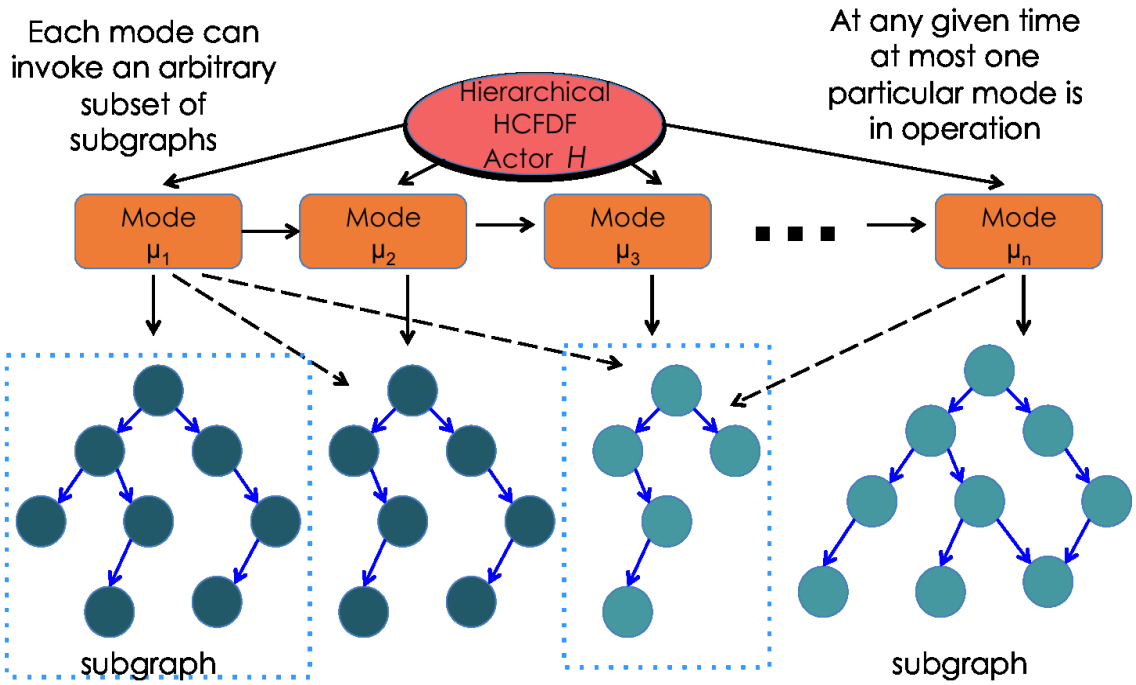


Figure 4.2: Modeling multi-mode DDDAS designs using HCFDF graphs.

execution within each $\mu_i$ can consist of concurrent executions of an arbitrary number of HCFDF-based subsystems (dataflow subgraphs), and parallelism can be exploited within and across these concurrently executing subsystems.

Additionally, in our proposed design environment, the $\mu_i$s can share HCFDF subgraphs among them to promote code reuse, and reduce program memory requirements. For example, if a common speech processing subsystem is invoked in multiple application modes, it can be referenced from each of those modes, while having separate parameter settings, if desired, across the different modes that employ it. This leads, for example, to a design representation of information fusion alternatives as parameterized subsets of dataflow subgraphs, where each subgraph can be specialized to a particular type of information source (e.g., image, video,

network event streams, speech, or high fidelity audio).

## 4.4 Dynamic, Multi-Mode Scheduling

To integrate system-level, dynamic, data-driven operation into the targeted class of signal processing applications, we describe in this section an adaptive scheduling strategy for dynamic configuration and scheduling of multi-mode HCFDF graphs. The scheduling approach described here is capable of dynamically adapting the selected application mode (e.g., high performance, high accuracy processing versus low energy, approximate processing) based on the overall health status of the target platform (e.g., available battery capacity), as well as on the data processing scenario (e.g., high-performance, alarm-driven scenarios versus low energy, standby scenarios).

The general scheduling approach, which we call the *DHMM* (*DDDAS-HCFDF Multi-Mode*) scheduler, involves a set of measurements $m_1, m_2, \ldots, m_k$ — from the target platform, operating environment or system output — that are to be taken at discrete times during execution. Here, each measurement $m_i$ corresponds to a distinct metric (e.g., instantaneous power consumption, remaining battery capacity, selected frequency content values for some kind of sensor data, or processing resource utilization as a percentage of available platform resources, to name a few possibilities).

A natural way to schedule these measurements is just after each iteration of the executing application mode, since *dataflow graph iteration* is a commonly

used concept of time window in the analysis of signal processing oriented dataflow programs (e.g., see [5]). Here, an application iteration can be defined to mean the processing period for a set of data frames (e.g., with one frame associated with each monitored sensor) for the current application mode, or can be parameterized to cover some number $F$ of frame sets, where $F$ can be adjusted dynamically to control to trade-offs among measurement overhead, adaptation frequency, and reactivity (the speed with which system reconfiguration can track changes in the measured data).

Figure 4.3 and Figure 4.11 show two design iterations (a simple version and an advanced version) of an example state machine $S_{DHMM}$ as part of the DHMM methodology.

The sequence of measurement vectors, $\{(m_1(i), m_2(i), \ldots, m_k(i)) \mid i = 1, 2, \ldots\}$, obtained by this application-iteration-level instrumentation process drives a state machine $S_{DHMM}$, where the states are in one-to-one correspondence with the application modes, and each state $\sigma$ has an associated function (i.e, a computational function, not just a mathematical function) $f_\sigma$. The purpose of each function $f_\sigma$ is to compute parameter values for the mode associated with the state $\sigma$ based on the newly observed instrumentation data (measurement vector), and any state variables that are maintained for $\sigma$. The state machine $S_{DHMM}$ thus plays a central role in relating the *instrumentation subsystem*, which generates the measurement vectors, to the available application modes and their underlying dataflow subgraphs.

The design of the instrumentation subsystem — including selection of the metrics $\{m_i\}$ — along with the design of the state machine $S_{DHMM}$ are important aspects of our overall adaptive scheduling methodology. The instrumentation subsystem and

$S_{DHMM}$ together with the HCFDF-based application- and mode-level dataflow subgraphs that they control lead to a precise, formally rooted, and platform-independent design framework for integrating DDDAS, dataflow, and multi-mode signal processing principles. In Section 4.5 we concretely demonstrate the DHMM scheduling methodology on a multi-modal, DDDAS-driven, design and implementation case study involving image processing.

We would like to emphasize that the objective of the DHMM scheduling methodology is not to introduce a new specialized scheduling algorithm for mapping dataflow graphs but rather to provide a systematic framework with which different schedules or scheduling algorithms can be integrated to provide DDDAS-driven, multi-mode integration for collections of signal processing subsystems (dataflow subgraphs). In particular, the "mode-level schedules" that are are used to execute specific application modes under specific mode parameter settings are *not* part of the DHMM framework specification. Such schedules can be derived by hand, statically by a software synthesis tool, at run-time or using a combination of synthesis-time and run-time techniques. Such separation of concerns between scheduling and system specification is a fundamental objective for dataflow-based signal processing environments (e.g., see [5]), and for model-based design tools in general.

## 4.5 Case Study: Face Detection

In this section, we demonstrate our proposed multi-mode, DDDAS-driven design approach, and our associated DHMM scheduling framework with a multi-mode

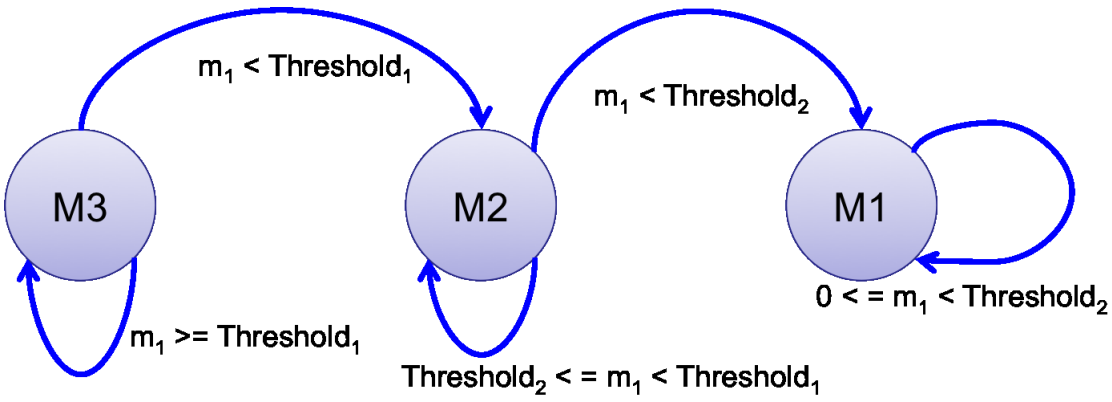


Figure 4.3: A simple $S_{DHMM}$ scheduler with one metric and 3 application modes.

application case study involving face detection. The measurement vector that we consider in the instrumentation subsystem consists of a single component $m_1$, which corresponds to battery capacity, and the state machine $S_{DHMM}$ is designed to gradually trade-off processing accuracy for energy efficiency as battery capacity drops from full to empty. Thus, we demonstrate how the targeted embedded system adapts in response to periodically measured data on system health, along with an underlying model of the design space across alternative classifier configurations.

The state machine design associated with this case study is illustrated in Figure 4.3.

### 4.5.1 Experimental Setup

Our experiments were performed through simulation on an Intel Core i7-2600K CPU (3.40GHz, 15GB RAM) running the Ubuntu 12.04 LTS operating system. The simulation — including HCFDF-based functionality for the DHMM scheduler, multi-mode application subsystems, and instrumentation subsystem — was imple-

mented using the LiD4E environment [34]. In particular, the C-based application programming interfaces (APIs) of LiD4E were employed; thus, our experimental system implementation can be viewed as a C language realization that employs LiD4E APIs to achieve the desired forms of high level dataflow semantics.

The experiments reported on in this section can be viewed as providing initial demonstration and validation of the multi-mode, DDDAS design methodology presented in this chapter. More complex experiments — e.g., involving multi-dimensional instrumentation spaces (measurement vectors with multiple components) and implementations on embedded platforms — are a useful direction for future work.

The face detection application that we experiment with in this chapter is based on an application introduced in [34], with modifications incorporated to integrate the DHMM scheduling framework with the metric $m_1$ described above for battery capacity, and a state machine $S_{DHMM}$ that is designed to provide decreasing levels of processing accuracy and energy consumption as the battery level decreases. These alternative accuracy/energy trade-offs are captured discretely through three separate states (application modes) in $S_{DHMM}$. The three modes correspond to three distinct classifier configurations, which can be viewed, respectively, as configurations that provide maximum energy efficiency; a trade-off among accuracy, energy efficiency, and false positive rates; and a minimum false positive rate. We refer to these modes as $M_1$, $M_2$, and $M_3$. Here, energy efficiency is measured in terms of the amount of energy consumed per classification operation (mode invocation). Thus, $M_1$ has the lowest energy consumption, $M_3$ has the highest, and $M_2$ has an

intermediate level of energy consumption.

The accuracy and false positive rates for a set of executed classification operations are defined, following standard convention, as follows. Suppose that $C$ classification tasks are performed, and among these, $c_1, c_2, c_3, c_4$ tasks represent the true positives, true negatives, false positives, and false negatives (see equation 4.1), respectively. Thus,

$$C = c_1 + c_2 + c_3 + c_4. \tag{4.1}$$

We define the associated classification *accuracy* as:

$$accuracy = (c_1 + c_2)/C \tag{4.2}$$

and the *false positive rate* (*FP rate*) as:

$$FPrate = c_3/C \tag{4.3}$$

In many kinds of operational scenarios — e.g., where false positives are much more costly compared to false negatives — the FP rate is viewed as being more important than maximizing accuracy (at least up to some allowable degradation in accuracy).

The scheduler state machine $S_{DHMM}$ is parameterized with a two-element vector, $\nu \in V$, called the *threshold vector*. Here, $V$, the set of permissible values for $\nu$, is defined by

$$V = \{(x, y) \mid 0 \leq y \leq x \leq 1\}. \tag{4.4}$$

Given an initial battery capacity $J$, transitions between modes are carried out in $S_{DHMM}$ by starting initially in $M_3$, transitioning to $M_2$ once the battery capacity falls below $x \times J$, and then transitioning to $M_1$ (the most energy efficient mode) once the battery capacity falls to $y \times J$. Certain boundary conditions in $V$ lead naturally to special cases in the trajectory of modes. For example, if $x = 1$, then we transition immediately to $M_2$, and if $x = y$, then $M_2$ is effectively skipped.

In our experiments, we use $F = 1$ as the iteration length (see Section 4.4), meaning that the DHMM scheduler makes its next assessment about whether to switch modes after each new image is processed. The SVM classifier parameters for all three application modes were developed using MATLAB, trained using the MIT CBCL face database [43], and then ported to C in the LiD4E environment.

### 4.5.2 Experiment Results

In our simulation setup, we estimate the energy consumption of a classification task as being proportional to the latency, and we assume that the target platform consumes negligible energy consumption during idle periods through use of power-saving sleep capabilities. More specifically, we assume a constant average power consumption $\rho$ across all modes so that the battery energy drained for a given mode invocation is estimated as $\rho \times \tau(\mu)$, where $\tau(\mu)$ is the average latency (processing time), as measured for mode $\mu$. This model is used to simulate draining of the bat-

tery from full capacity to empty capacity. This simulated draining process in turn creates a stream of battery capacity data, which is used to drive the DHMM adaptation process implemented by $S_{DHMM}$. This is a relatively simple model of energy consumption; an application of a more sophisticated energy models is discussed in Section 4.6.1.

Figure 4.4 through Figure 4.6 show experimental results for several different threshold vectors. The overall energy efficiency (Figure 4.4) gives a measure of the total number of processed images across the lifetime of the system (i.e., until the battery is fully drained). The three threshold vectors towards the right (labeled SDF1, SDF2, and SDF3) each correspond to execution of a single mode for the entire input stream (no state transitions). Such implementations represent statically-structured implementations that do not employ multi-mode/DDDAS capabilities, and can be implemented as synchronous dataflow (SDF) graphs, without use of more dynamic features, including HCFDF modeling or the proposed DHMM scheduler.

Intuitively, the DHMM-based system provides a way to achieve configurable, graceful degradation of classification quality (accuracy and FP rate) as the battery expires. This can be important, for example, if a mission lasts significantly longer than expected. The results in Figure 4.4 through Figure 4.6 help to quantify this kind of graceful degradation, and also demonstrate another important advantage of the DHMM-based approach: the approach allows for finer-grained control over the overall design evaluation space (i.e., in this case, the space involving energy efficiency, accuracy, and FP rate). By varying the threshold vector, the designer or a run-time system can steer the overall system performance (across the entire mission) towards

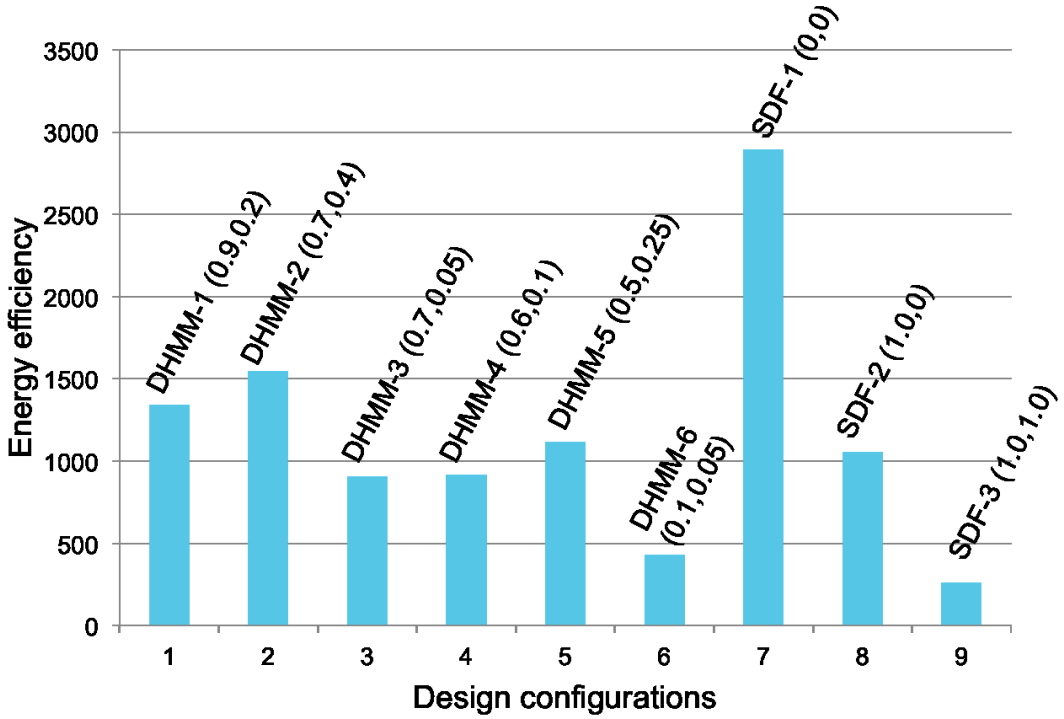


Figure 4.4: Energy efficiency trends across DHMM and static designs for a simple $S_{DHMM}$.

a specific Pareto-optimal point in the space that represents the best trade-off for the application. Thus, rather than being confined by just the trade-offs provided by the individual classifiers (i.e., the three rightmost bars in Figure 4.4 through Figure 4.6 for this case study), the designer or run-time system has a large amount of control in steering the overall performance into other regions of the underlying design evaluation space. These capabilities — configurable and graceful degradation and the production of new, Pareto-optimal operating alternatives — represent significant advantages derived by applying the multi-mode, DDDAS techniques proposed in this chapter.

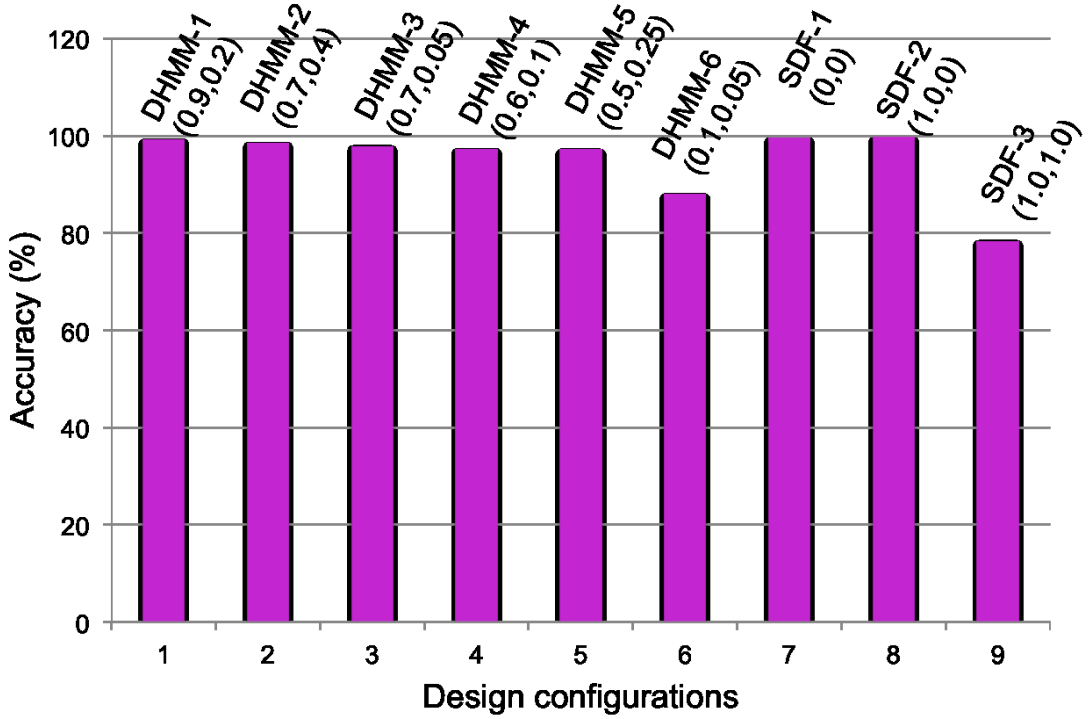


Figure 4.5: Average accuracy trends across DHMM and static designs for a simple $S_{DHMM}$.

## 4.6 Additional Experiments

In this section, we discuss additional experiments that provide further demonstration of the methods introduced in this chapter.

### 4.6.1 Energy Consumption Model

One of the key challenges of deploying adaptive stream mining systems on distributed, embedded environments is to carefully incorporate the energy consumption of the embedded platform into design space exploration processes. Such distributed

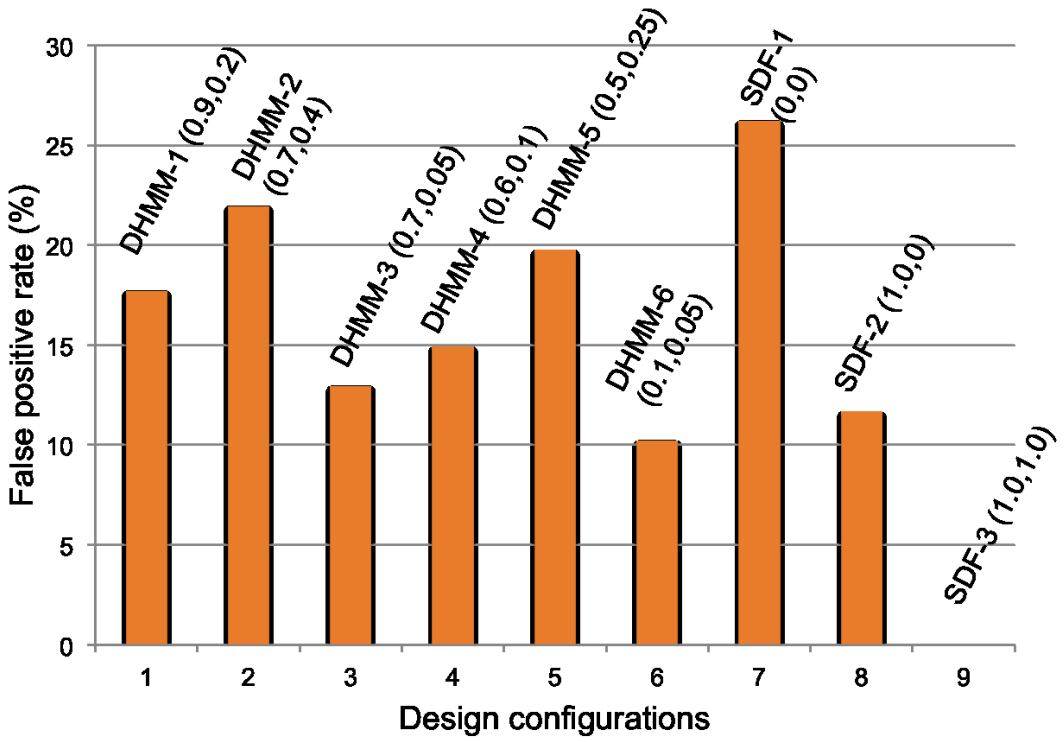


Figure 4.6: False positive rate trends across DHMM and static designs for a simple $S_{DHMM}$.

embedded platforms often operate from highly limited energy sources. In this section, we have carefully integrated energy consumption into design space exploration for stream mining system design. The energy model we use in this section is based on the model described in [54] with a number of adaptations to make it better suited to the targeted class of applications and platforms in this work. The goal of this section is not to introduce a novel energy consumption model but to support systematic integration of designer-specified energy models into design space exploration for adaptive stream mining systems.

When the energy model described in [54] is applied to a given execution of a

dataflow-based application graph, each dataflow actor $\nu$ is characterized by a pair of attributes, $(P(\nu), t_i(\nu))$, where $P(\nu)$ denotes an estimate of the average power consumed by the actor and $t_i(\nu)$ denotes the $i$th execution time instance — that is, the execution time taken by the $i$th firing of $\nu$. Similarly, each dataflow edge $e$ is characterized by a pair of attributes $(P(e), t_i(e))$, that represent, respectively, the average power consumption for data communication across the edge, and the time taken by the $i$th data transfer across the edge.

Utilizing these attributes, the total energy consumption for an application graph execution can be estimated as:

$$E_{total} = \sum_{\forall \nu \in V} ( \sum_{i=1}^{firings(\nu)} P(\nu) \cdot t_i(\nu)) + \sum_{\forall e \in E} ( \sum_{i=1}^{comm(e)} P(e) \cdot t_i(e)). \tag{4.5}$$

For more details on the energy model that we have applied, we refer the reader to [54]. We adapt this model in this section by using it to derive estimates of average power consumption for each application mode in a multi-mode adaptive stream mining system.

### 4.6.2 Execution Time Model

The experiments conducted in this section of the thesis are simulations based on offline measurements of actor and embedded platform characteristics. Techniques related to execution time modeling and analysis have been studied extensively in the literature. These techniques can be broadly classified in to static, probabilistic, and hybrid methods. Static methods that estimate the *worst case execution time*

(*WCET*) are covered, for example, in [55–57]. However, since our design methods in this thesis make extensive use of dynamic system-level adaptation, existing static execution time models and analysis methods do not match our design framework well.

Instead, we adapt in our work the empirical execution time modeling approach proposed in [58]. Using this approach, we derive execution time probability distributions through actual measurements on the targeted hardware and across the set of relevant application modes. The model proposed in [58] utilizes measured time durations for response time and round-trip time associated with computational tasks, along with measurements to model "null-code". This "null-code" component of the estimation method is used to model overheads, including communication cost.

Motivated by the modeling approach proposed in [58], we measure classification time and total execution time on targeted stream mining platforms utilizing the software development kits (SDKs) provided by the platforms. From this collected execution time data, we compute the marginal probability distribution (M-PDF) for each relevant application task, and apply the resulting M-PDF to model execution time when simulating the performance of system implementations for the associated application and platform. In particular, during simulation, we randomly generate an execution time value for each relevant actor firing or subsystem execution using a corresponding M-PDF that is generated based on the collected execution time data and M-PDF derivation process described above.

Further details on our employed execution time modeling approach are discussed in Section 4.6.3. Figure 4.7 through Figure 4.10 show the M-PDF distribution

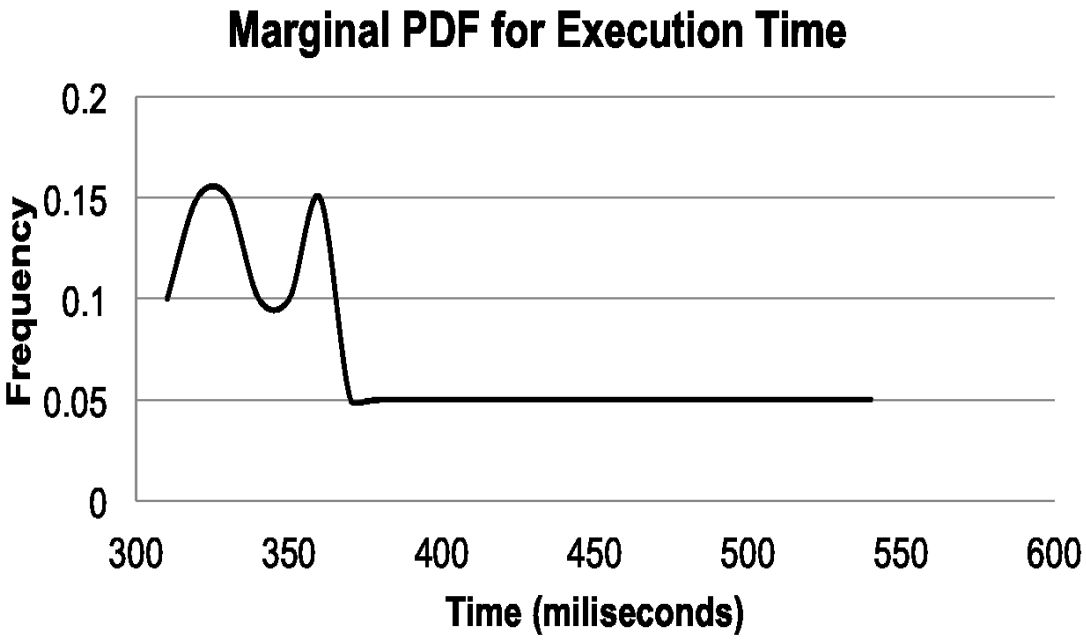


Figure 4.7: Marginal execution time probability distribution for Classifier 1.

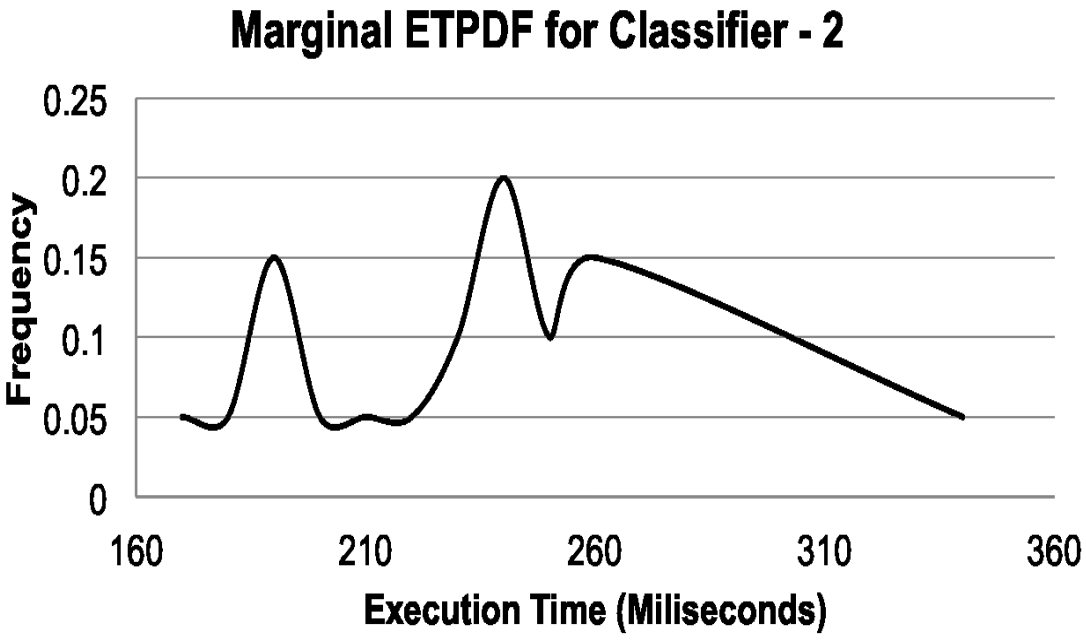


Figure 4.8: Marginal execution time probability distribution for Classifier 2.

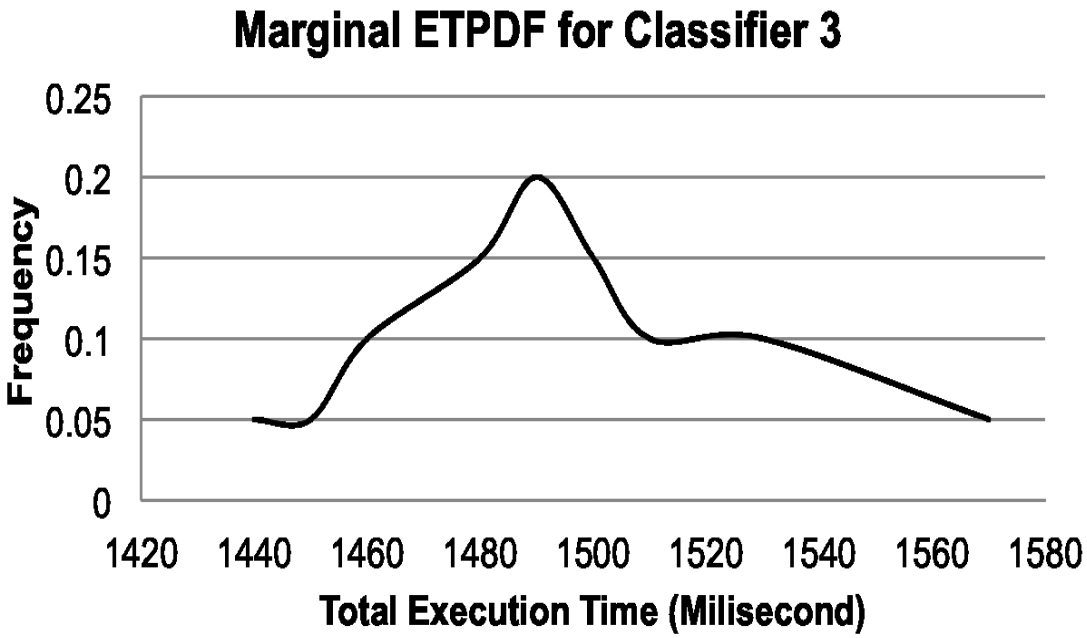


Figure 4.9: Marginal execution time probability distribution for Classifier 3.

for each classifier subgraph.

### 4.6.3 Experimental Setup

We utilized a *Nexus 7*, Android-based mobile platform as our primary target platform for the system implementation demonstrated in these experiments. The execution time instances were generated by porting the dataflow-based C language implementation (LIDE-C implementation) from a PC environment (where the application was initially prototyped) to the SDK of the Nexus 7 platform.

A minimum of 20 execution time instances for each classifier configuration was generated. The marginal-PDF was then computed based on the Execution Time Probability Density Function (ETPDF) model introduced in [58] and described in

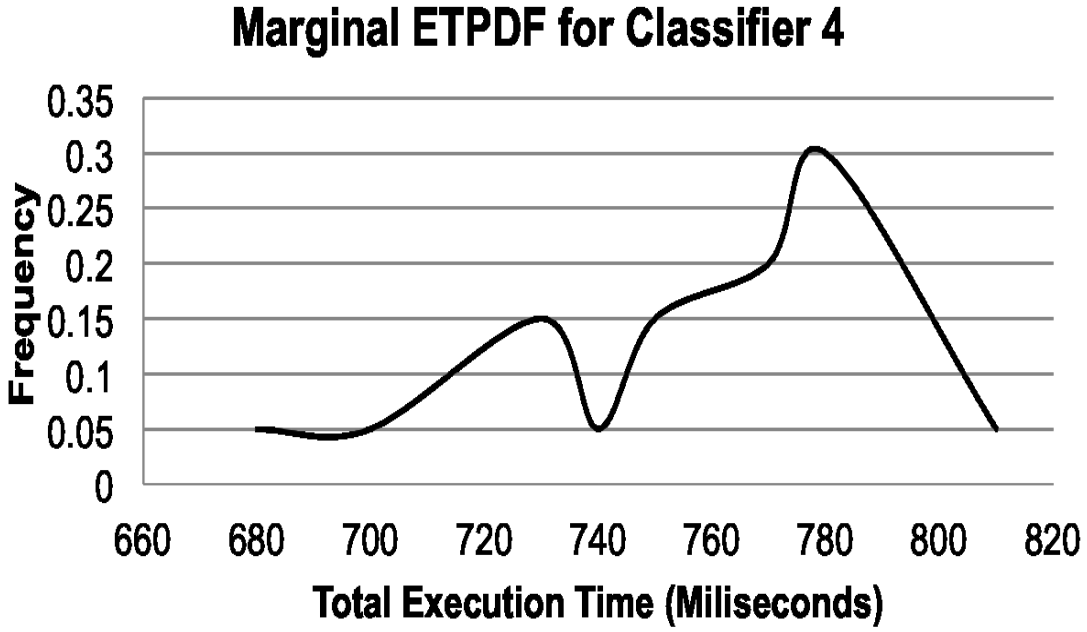


Figure 4.10: Marginal execution time probability distribution for Classifier 4.

Section 4.6.2. In addition to the classifier modes $M_1$, $M_2$, and $M_3$ described in Section 4.5.1, we developed a new classifier configuration $M_4$. This new classifier mode can be considered as having operational trade-offs that are intermediate with respect to $M_2$ and $M_3$ in terms of the investigated metrics.

Based on our empirical approach to execution time and energy modeling, the energy consumption of the four classifiers can be ranked in the following order (from highest to lowest energy consumption): $M_3$ (highest), $M_4$, $M_2$, $M_1$ (lowest).

The power consumed by the different classifier configurations was modeled as described in Section 4.5, along with the execution time models from the derived marginal ETPDFs that characterize classifier run-time efficiency. The addition of the new classifier configuration ($M_4$) enabled more control capability in the state

machine $S_{DHMM}$ with increased flexibility in exploring operational trade-offs. This flexibility came at the expense of a more complex state machine design.

### 4.6.4 Measurement Vector and Performance Evaluation Points

We employed a 2-dimensional measurement vector $(m_1, m_2)$, where $m_1$ represents battery capacity (as before), and $m_2$ is a measure of system throughput.

We also introduce a system input called the *performance evaluation point* (PEP), which is introduced so that a user or automated supervisory engine can control the frequency with which periodic system performance can be evaluated. Thus, the PEP can be varied, for example, to trade-off accuracy, energy efficiency, and real-time performance in different ways.

In our experimental setup we covered both externally generated (e.g., user-specified) and automated PEP configuration of the system. Here, the user specified PEP is modeled as a Poisson process with independent identically distributed (iid) exponential inter-arrival times. The expected value $\lambda$ was varied for each DHMM experiment shown in Figure 4.12 through Figure 4.15. The rate value, $\lambda$, is derived from the number of control points in the DHMM state machine $S_{DHMM}$ .

Automated PEP configuration was modeled through a uniform distribution. Such automated PEP configuration allows system performance to be assessed consistently, and independent of any external performance assessment control.

The approaches employed here for PEP configuration are relatively simple. Investigation of more advanced PEP approaches for DHMM execution is an interesting

direction for future work.

For more robust system operation, we introduced a *power down state* in the DHMM state machine $S_{DHMM}$. The power down state is employed to gracefully power down the system once the battery level falls below a pre-defined threshold. This threshold is represented by $\epsilon$. Thus, a state transition to the power down state is made once the battery capacity measurement indicates that $m_1 \leq \epsilon$.

### 4.6.5 Experimental Results

The extended DHMM state machine $S_{DHMM}$, based on the enhancements described above, is shown in Figure 4.11. Here, functions $f_1(m_1)$ and $f_2(m_1)$ provide Boolean indicators of whether or not the system is over-performing or under-performing, respectively, in relation to the corresponding thresholds defined for metric $m_1$. Similarly, $f_3(m_2)$ and $f_4(m_2)$ represent indicators of over-performance and under-performance, respectively, in relation to pre-defined thresholds for metric $m_2$. The functions $g_1(\cdot)$ and $g_2(\cdot)$ represent control triggers that correspond to automatically-generated and externally-generated, PEP-related transition points, respectively.

As mentioned in Section 4.6.3, automated PEP execution was modeled in our experiments using a uniform distribution and externally-generated PEP execution was modeled using using a Poisson arrival process. The battery capacity threshold $\epsilon$ for the power down state was set to 3.5% of the total battery capacity. The application mode states that are in one-to-one correspondence with the classifier

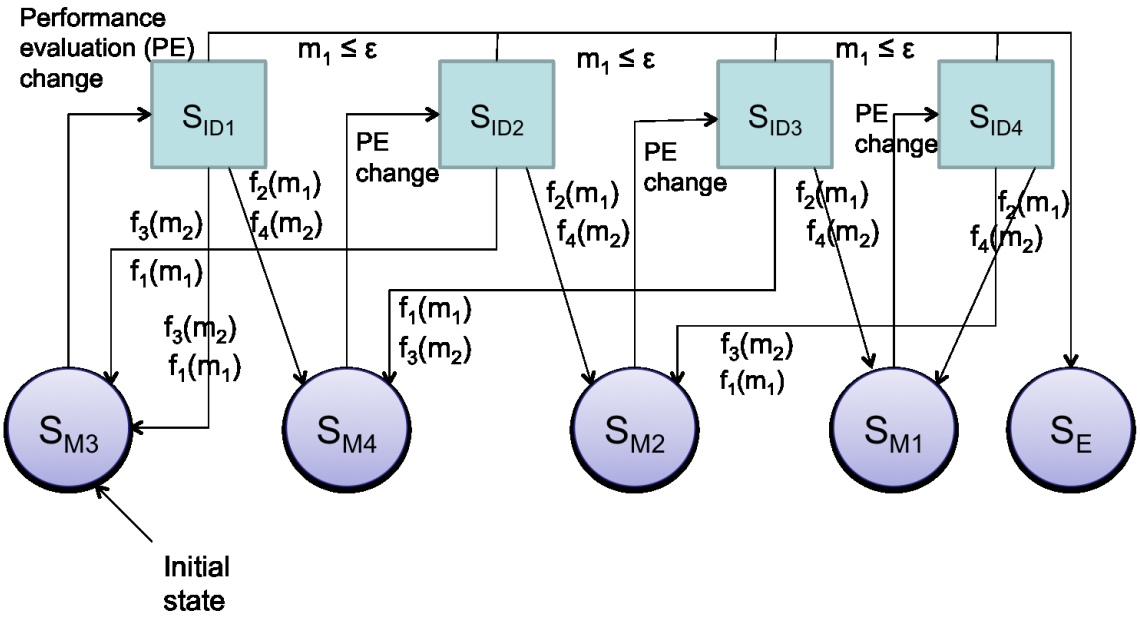


Figure 4.11: An extended version of the DHMM state machine for our face detection case study.

subsystems are labeled $S_{Mi}$, where $i$ ranges from 1 through 4 and provides the index of the corresponding classifier configuration. Thus, $i = 1, 2, 3, 4$ correspond to application modes $M_1, M_2, M_3, M_4$, respectively. Intermediate states that provide finer granularity of control (at a lower level of granularity compared to being restricted to only mode-level control) are denoted as $S_{IDi}$, where $i$ provides the index of the intermediate state.

Figure 4.12 through Figure 4.15 show experimental results for the extended state machine setup described above. We experimented with 7 different DHMM configurations. In addition to the static configurations that we employed in the previous experiment, we also added a new static-configuration experiment corresponding to the new classifier configuration $M_4$ (this is labeled as SDF–4 in Figure 4.12 through

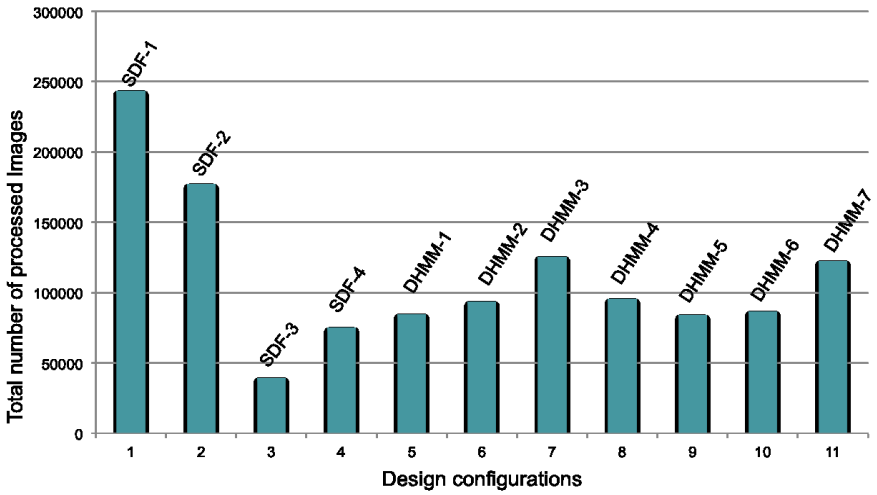


Figure 4.12: Energy efficiency trends across DHMM and static systems for our extended $S_{DHMM}$ design.

Figure 4.15).

Figure 4.13 shows the overall deadline miss rate of the system for each DHMM and static (SDF) system experiment.

## 4.7 Summary

In this chapter, we have developed an approach to design and implementation of multi-mode, data driven signal processing systems. We have developed methods for modeling and designing such systems using integrated principles of dynamic data driven application systems (DDDAS) and high-level, dynamic dataflow models of computation. We have introduced a scheduling framework, called the DHMM (DDDAS-HCFDF Multi-Mode) scheduler, for instrumentation-driven, adaptive scheduling in multi-mode signal processing systems. Through a case study

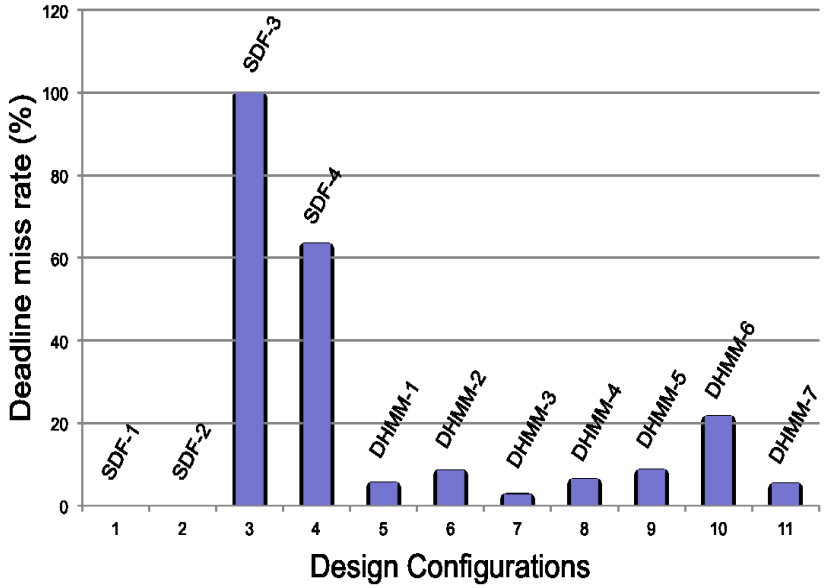


Figure 4.13: Deadline miss rate trends across DHMM and static systems four our extended $S_{DHMM}$ design.

of an energy-constrained, multi-mode face detection system, we have demonstrated and quantified significant advantages of our proposed new methods. Useful directions for future work include (1) extensions to multiple sensing modalities, such as integrated image and speech processing, and (2) experimentation with instrumentation subsystems that produce multidimensional outputs (e.g., channel quality in addition to power consumption).

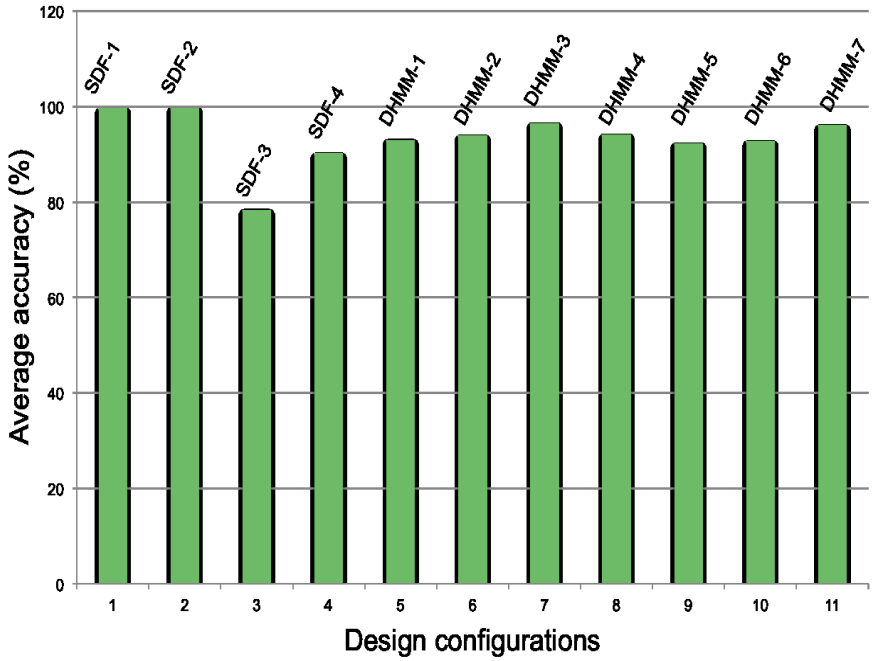


Figure 4.14: Average accuracy trends across DHMM and static systems for our extended $S_{DHMM}$ design.

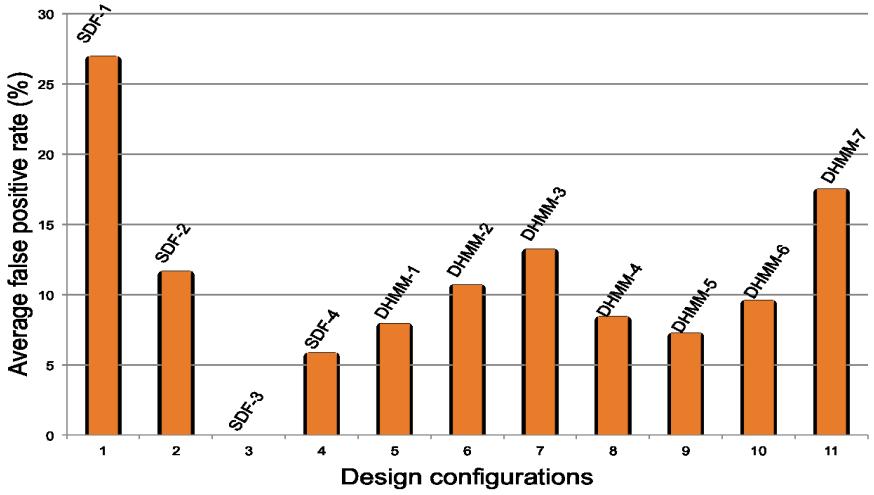


Figure 4.15: False positive rate trends across DHMM and static systems for our extended $S_{DHMM}$ design.

# Chapter 5: Multiobjective Design Optimization

## 5.1 Introduction

A challenging aspect of adaptive stream mining (ASM) systems is the diverse sets of operational constraints and objectives under which they must be deployed. The specific constraints that take precedence may depend strongly on the operational scenario and associated data. For example, in the midst of a security breach or intrusion, conserving energy is less important while delay is key; on the other hand, if there is no detected threat, conserving energy may be critical. Furthermore, such multidimensional constraints and objectives are often in conflict with one another so that trade-offs must be carefully guided and rigorously optimized to achieve results that yield acceptable levels of reliability and quality of service. For example, classification accuracy (the rate of correct classifications), false positive rates in classification, processing latency, processing throughput, and energy consumption per classification operation are metrics that may all be relevant to some degree in a particular stream mining deployment. Conventional approaches to design and implementation of ASM systems often focus on small subsets of relevant metrics in isolation (e.g., the trade-off between accuracy and false positive rate) or orient the implementation process toward a particular subspace (e.g., throughput-constrained

accuracy maximization).

Motivated by these complex, multidimensional, data-dependent design spaces in ASM systems, we demonstrate in this chapter methods for integrated modeling and multiobjective design optimization of real-time stream mining systems. Our proposed design framework is readily adaptable to different kinds of operational constraints and objectives. For concreteness, we develop our methods in the context of real-time performance, multidimensional stream mining performance (precision and recall), and energy efficiency. These metrics are discussed in detail in Section 5.4. Using a design methodology centered on data-driven control of and coordination between alternative dataflow subsystems for stream mining (classification modes), we develop systematic methods for exploring complex, multidimensional design spaces associated with dynamic stream mining systems, and deriving sets of Pareto-optimal system configurations that can be switched among based on data characteristics and operating constraints.

We demonstrate and experiment with our methods for data-driven, multiobjective optimization through their integration in the *Lightweight Dataflow for Dynamic Data-Driven Application Systems Environment* (LiD4E), which is a software tool for experimentation with and optimization of dataflow-based design methods for ASM systems [34], as described in Chapter 3. Using LiD4E together with our new methods for multiobjective optimization, we experiment with a multiclass vehicle classification system that categorizes vehicles among three distinct classes — cars, buses and vans — from images. Through experiments on this vehicle classification application, we demonstrate the effectiveness of our methods in deriving

Pareto-optimal design options and quantifying complex implementation trade-offs. These capabilities can provide significant insight to the system designer to identify the set of design configurations that best matches the targeted set of application scenarios and their associated system requirements.

The remainder of this chapter is organized as follows. In Section 5.2, we discuss related work in DDDAS methods, real-time stream mining, and dataflow-based design methodologies for signal processing systems. Section 6.3 introduces our proposed new multiobjective design optimization framework, and Section 5.4 presents a vehicle classification case study to demonstrate the framework. In Section 5.5, we present experimental results from this case study. Finally, Section 5.6 provides conclusions and future research directions.

Material in this chapter was published in preliminary form in [59].

## 5.2 Related Work

The work presented in this chapter is rooted in core concepts of the DDDAS paradigm [7]; real-time stream mining [15]; and dataflow-based design methodologies for signal processing systems (e.g., see [4,5]). In this form of dataflow modeling, applications are represented in terms of *dataflow graphs*, where graph vertices (*actors*) represent signal processing tasks of arbitrary complexity, and edges represent logical FIFO communication channels between pairs of actors. In this chapter, we apply dataflow as a programming model with semantics that are matched to the domain of adaptive stream mining systems [34, 44]. This modeling approach differs

from uses of dataflow as a compiler intermediate representation (e.g., see [45]), and as a form of computer architecture [46].

The work presented in this chapter builds upon our previous work on ASMs for multimedia applications [34] that is described in Chapter 3, and extends the dynamic, multi-mode stream mining framework [44] that is discussed in chapter 4 with powerful capabilities for multiobjective design space exploration and optimization. More specifically, contributions introduced in this chapter include new techniques for modeling, control and optimization of multiobjective design spaces in ASM implementation; extension of the multi-mode design framework of [44] to multiclass recognition systems (i.e., to classifiers that map to two or more different output classes); and application to a multiclass vehicle detection problem that is relevant to surveillance and traffic monitoring.

Design and implementation techniques for stream mining systems have been studied before in a statically configured environment, and with relatively fine granularity (low level) optimizations on application performance (e.g., see [47–49]). Here, by "statically-configured," we mean that the processing methods are not adapted dynamically in response to data characteristics or operational context. Our work in this chapter deviates from this body of prior work in that our focus is on a dynamic, data-driven implementation context, and also, we focus on coarser granularity optimizations — in particular, optimizations for configuring and coordinating across different stream mining classification subsystems and application modes.

Incorporation of data-driven operation in individual signal processing functional components has been studied in [50, 51]. This related work has been de-

veloped in the context of speech recognition. Although this work relates to the dynamic, data-driven theme of our contribution in this thesis, the approach that we demonstrate in this chapter is more flexible in terms of data-driven operation since we consider adaptation of application modes globally (at the dataflow graph and scheduling levels) as well as locally (at the level of individual actors or subsystems). In contrast, this body of related work on data-driven speech processing focuses on local optimizations. However, techniques derived from works such as [50, 51] can provide useful building blocks (parameterized actor and subsystem designs) for the DDDAS design framework described in this chapter. Integration of such building blocks into our proposed framework is a useful direction for future work.

We emphasize that the objective of this chapter is not to introduce new types of classification techniques nor to endorse a particular type of classifier, but rather to provide a systematic framework for optimized configuration, control, and coordination across arbitrary sets of complementary classifiers (i.e., classifiers with complementary profiles of operational trade-offs). In our implementations and experiments, we utilize Support Vector Machine (SVM) classifiers, although our design framework is readily adaptable to the use of other types of classifiers. Use of SVM classifiers for low-sample data sets, and as efficient, robust components for general classification purposes has been motivated extensively in the literature (e.g., see [60, 61]).

## 5.3 Design Methodology

In this section, we introduce the system model that we employ in our new multiobjective design optimization framework, which we refer to as the *ASM multiobjective design optimization framework*, abbreviated as AMDO. AMDO is built upon the DDDAS-HCFDF-Multi-Mode (DHMM) scheduling framework introduced in [44] and described in Chapter 4. Here, HCFDF stands for hierarchical core functional dataflow [34], which is the underlying model of computation for the DHMM framework that was introduced in Chapter 3.

### 5.3.1 Background on DHMM

We first review in this section key aspects of the DHMM system model that are inherited by AMDO. The developments in this chapter build on the DHMM model, and incorporate flexible and powerful new capabilities for multiobjective optimization and design space exploration. In DHMM, an ASM system design is represented as a set of mutually exclusive *application modes*:

$$S_M = \{\mu_1, \mu_2, \ldots, \mu_N\}. \tag{5.1}$$

.

Here, each $\mu_i$ represents a set of application subsystems that are active during the corresponding mode together with the configurations, such as actor-, application- and schedule-level parameters, that are to be applied to the subsystems whenever $\mu_i$

executes. Each design is also characterized by a set of measurements, corresponding to the associated DDDAS-based instrumentation subsystem, $M = m_1, m_2, \ldots, m_k$. These measurements can be made from arbitrary sources, including the system input, target platform, system output or operating environment. Each $m_i$ corresponds to a distinct metric, such as power consumption, remaining battery capacity, or selected frequency content profiles for some kind of sensor data.

A key aspect of the DHMM model is a state machine $S_{DHMM}$ in which states correspond to application modes, and transitions correspond to changes made by the executing system to the current mode in response to input data that is monitored by $S_{DHMM}$. This input data comes from the measurements $m_i$, which are performed iteratively according to periodic processes or other kinds of timing patterns (e.g., dependent on the current mode).

In DHMM, the functionality of specific application modes is represented using the hierarchical core functional dataflow (HCFDF) model of computation [34], while $S_{DHMM}$ is employed for dynamic and adaptive model-based coordination and parameter control across different modes. In HCFDF-based dataflow graph specifications, software components (actors) are specified in terms of sets of processing *modes*, where each mode has static *dataflow rates* — i.e., each mode produces and consumes a fixed number of data values (tokens) on each actor port. However, different modes of the same actor can have different dataflow rates, and the actor mode can change from one actor execution (firing) to the next, thereby allowing for dynamic dataflow behavior (dynamic rates). Additionally, HCFDF allows dataflow graphs to be hierarchically embedded within actors of higher level HCFDF graphs,

thereby allowing complex systems to be constructed and analyzed in a scalable manner. For further details on the HCFDF model of computation, we refer the reader to [34] and Chapter 3.

### 5.3.2 AMDO Design Methodology

The AMDO design methodology incorporates the instrumentation subsystem $M$ and mode-transition state machine $S_{DHMM}$ of DHMM. The methodology additionally incorporates a parameterization $P$ of $S_{DHMM}$ for use in exploring the design space associated with implementations that are controlled by $S_{DHMM}$ in conjunction with the underlying application modes. More specifically, $P = (p_1, p_2, \ldots, p_K)$, called the *design space parameter set* (*DSPS*) of the AMDO model, is a sequence of parameters of $S_{DHMM}$, where each $p_i$ has an associated domain $domain(i)$, which gives the set of admissible parameter settings (*configurations*) for $p_i$ during execution of $S_{DHMM}$. For clarity and conciseness, we assume in the remainder of this chapter that the $domain(i) \subset \mathbb{R}$ for all $i$, where $\mathbb{R}$ denotes the set of real numbers.

Figure 5.1 shows an overview of the AMDO design methodology. The AMDO framework is designed to systematically integrate designer-specified sets of optimization objectives, explore the resulting multidimensional design spaces and extract Pareto-optimal design configurations.

When applying the AMDO methodology, the parameterization $P$ of $S_{DHMM}$ is central to the processes of design space exploration and multiobjective optimization. Different parameter configurations of $S_{DHMM}$ in general lead to different ways

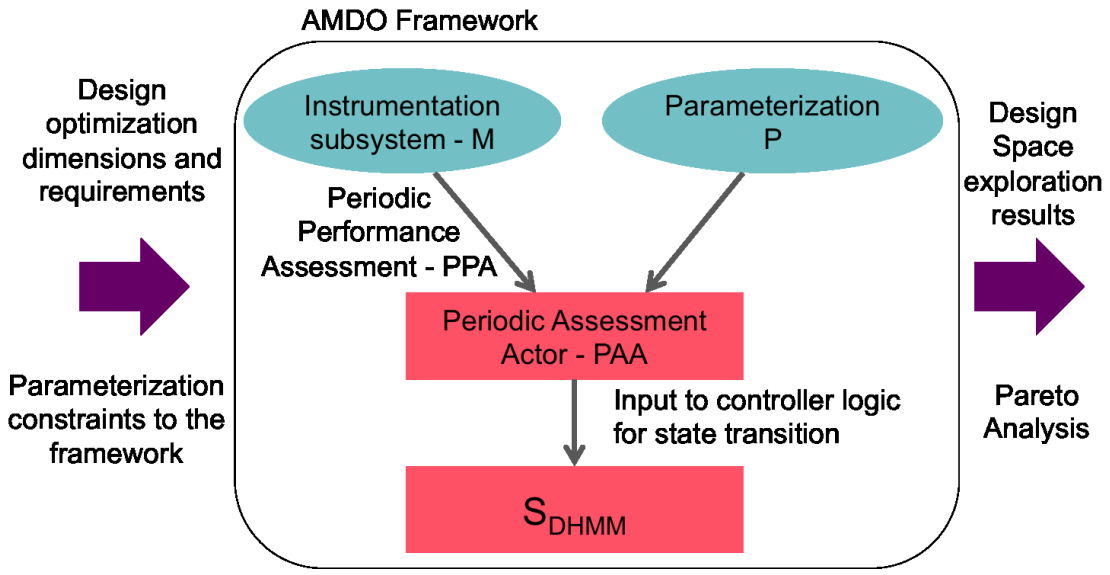


Figure 5.1: An overview of the adaptive multiobjective design optimization (AMDO) framework.

in which data-driven adaptation is controlled, and in which the multidimensional design evaluation metrics, such as energy consumption, real-time performance, and stream mining accuracy, are traded-off throughout the execution process. Additionally, the high level dataflow model of the targeted ASM application together with the FSM-driven application governed by $S_{DHMM}$ provides a model-based representation that can be employed for efficient simulation so that a wide variety of alternative parameter configurations and associated design points can be evaluated.

Two other aspects in the operation of an AMDO-based stream mining implementation are periodic performance assessment (PPA), and performance assessment actors (PAAs). In each state of $S_{DHMM}$, the recent performance of the system is assessed in terms of the set of relevant metrics $M$. This PPA process helps to determine whether $S_{DHMM}$ should remain in its current state or whether a transition

should be made to a different state. The operation of the PAAs may in general depend on the values of parameters in $P$. The determination of whether or not a transition is made and which new state should be the target of each PPA-related transition is made by $S_{DHMM}$ with input from the PAAs. Each PAA $A$ is a software component (dataflow actor) that takes as input a selected subset of data obtained from the DDDAS instrumentation subsystem during a window of recent operation (e.g., during the last 10ms or last 100 processed data packets).

On each execution of $A$, the output of $A$ is a member of the set

$$s_{PAA} = \{\gamma_o, \gamma_i, \gamma_u\}, \tag{5.2}$$

where $\gamma_o$ represents an indication by $A$ that the system is currently overperforming with respect to the form of performance assessment carried out by $A$. Similarly, $\gamma_u$ represents and indication by $A$ that the system is underperforming, and $\gamma_i$ indicates that the performance of the system is within an intermediate range — neither too high (at potential expense of other objectives) nor too low. Intuitively, a PAA can be viewed as a standard interface for capturing data-dependent characteristics of system operation, and relating them dynamically to a compact set of values ($\gamma_o$, $\gamma_i$, and $\gamma_u$). The values generated by the different PAAs can then be processed in an integrated way by $S_{DHMM}$ to control overall system operation.

For example, an AMDO system could be designed with three PAAs $A_1, A_2, A_3$ that correspond, respectively, to performance assessment for speech processing quality (accuracy), energy consumption, and processing speed. During each PPA, these

PAAs would each provide an input to the controller for $S_{DHMM}$ indicating the "health" of the system's recent performance with respect to the corresponding assessment considerations. Logic within the controller would then process these inputs to determine whether or not to remain in the current state, and what state to transition to if a transition is to be made. For example, if the system is found to be underperforming in terms of energy consumption (i.e., consuming excessive amounts of energy), this may favor a transition to a more energy-efficient application mode. Similarly, overperforming with respect to speed may lead to transition to a processing mode that is slower and more favorable in terms of other objectives, such as energy consumption or quality.

As with the state machine parameterization $P$, the design of the PAAs, and the associated controller logic for processing the PAA outputs are design issues of the given AMDO. The objective of the AMDO design methodology is thus to raise the level of abstraction for stream mining system implementation in a structured manner so that the system designer can focus on a standard, well-defined set of DDDAS-based system components — $S_{DHMM}$, $P$, the PAA set — that interact in a systematic manner.

Thus, we represent an AMDO system $\alpha$ by a tuple

$$\alpha = (S_{DHMM}, P, T), \tag{5.3}$$

where the elements of this tuple respectively specify the state machine, parameterization, and PAA set associated with $\alpha$.

Using an AMDO system $\alpha = (S_{DHMM}, P, T)$, the designer can evaluate multidimensional system performance for a variety of parameter settings within $P$ to generate alternative design points, while each parameter setting influences system operation (through $S_{DHMM}$ and $T$) to trade off different performance objectives in a specific way. In Section 5.4 and Section 5.5, we demonstrate the application of the AMDO design methodology on a practical surveillance application case study involving vehicle detection. This case study helps to make the developments in this section more concrete, and to demonstrate the utility of the AMDO methodology as a framework for multiobjective design space exploration and optimization of ASM systems.

## 5.4 Case Study: Vehicle Classification

To validate and demonstrate the AMDO framework, we have developed a multiobjective optimization case study of a data-driven ASM application that is relevant to surveillance systems. Specifically, our case study involves a vehicle classification system in which images of detected vehicles are analyzed to classify each vehicle as either a bus, car or van. The classification system is assumed to be a mobile system that is capable of being deployed with agility and low cost in operational environments. This mobile deployment feature makes energy efficiency an important metric to consider in the design evaluation space for the system. Figure 5.3 shows an illustration of the vehicle classification system. Section 5.5.2 describes the application in more detail.

We have performed extensive simulations to evaluate a complex, five-dimensional design evaluation space (i.e., a space of trade-offs involving selected implementation metrics) that is based on several relevant, and often competing deployment objectives. Specifically, the design evaluation space considered encompasses the metrics of throughput (data rate), deadline miss rate (real-time performance), energy efficiency, precision for detecting cars (one objective related to classification accuracy), and recall for detecting cars (another accuracy-related objective). Thus, a main goal of the case study is to expose Pareto points in a complex multidimensional space of designs for deploying the vehicle classification application on a targeted mobile device.

Here, the throughput, in terms of images per second, gives the rate at which the system can process images. If $T$ denotes the throughput, then the reciprocal $(1/T)$ specifies the *deadline*, which is in units of seconds per image, and gives the maximum time allowed to process a single image. Whenever the AMDO system fails to process an image within its associated deadline period, a deadline miss occurs. For a given stream mining execution consisting of an input stream that contains $I$ images, the deadline miss rate $r$ is computed as:

$$\text{deadlinemissrate}(r) = (N_{miss}/I), \tag{5.4}$$

where $N_{miss}$ is the total number of deadline misses encountered throughout the execution.

Figure 5.2 provides an illustration of the state machine $S_{DHMM}$ for our AMDO-

based vehicle classification system. The state machine includes three application modes, labeled $Z_{M,1}, Z_{M,2}, Z_{M,3}$, which represent one-against-one (1A1) support vector machine (SVM) classifier subsystems with varied parameter configurations. For background on this type of classifier, we refer the reader to [60]. The classifiers are configured with Gaussian radial basis function kernels that have different combinations of sigma and box constraint values. These three alternative application modes yield different operational trade-offs in terms of execution time, energy consumption, precision, and recall. The AMDO framework provides a systematic way to exploit such variety in application modes to derive diverse sets of alternative design points (Pareto designs) during multiobjective optimization. The states in Figure 5.2 with labels of the form $Z_{P,i}$ correspond to PPA points. Each of these states encapsulates a single PAA, and is entered periodically from its associated application mode. The transitions in the state machine are executed either from periodic interrupts that trigger PPAs or from decisions that are computed from the relevant PAAs.

The state labeled $Z_{M,E}$ in Figure 5.2 is a special state that is dedicated to providing graceful shutdown of the system once the battery capacity $c$ has fallen to a value that is less than or equal to a pre-defined threshold $\epsilon$. In our experiments (see Section 5.5), we employed $\epsilon = 5\%$.

Since our targeted application is a multiclass classification application (a classification application that involves more than two classes), we employ *precision* and *recall* as metrics for assessing classification accuracy. These metrics are commonly used for multiclass classification systems. We arbitrarily choose cars as the relevant vehicle class for the precision and recall calculations. Thus, the precision is calcu-

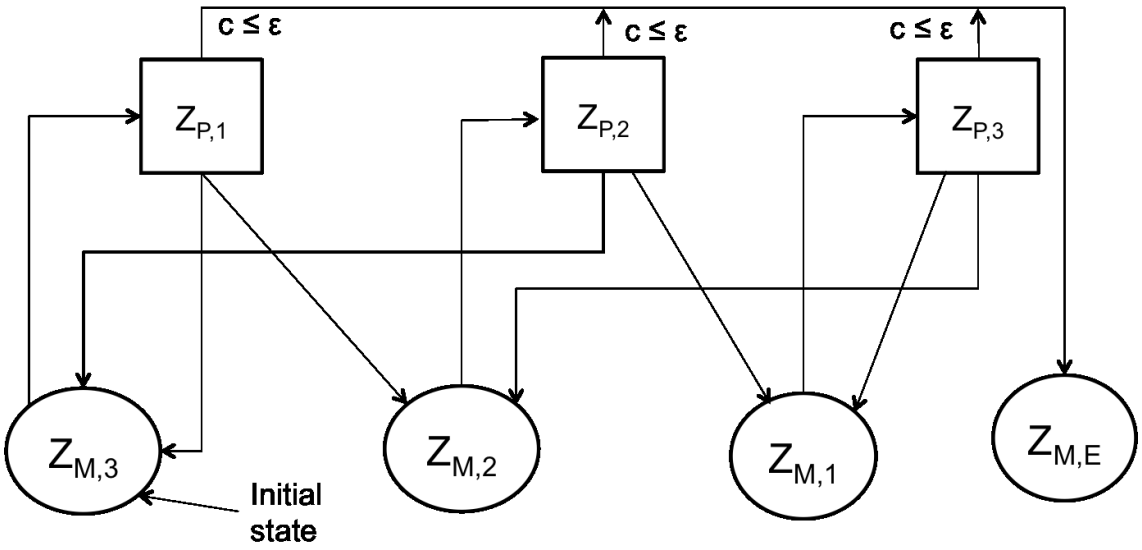


Figure 5.2: An illustration of $S_{DHMM}$ for the experimental vehicle classification system.

lated as $TP/(TP + FP)$ and the recall is calculated as $TP/(TP + FN)$, where $TP$, $FP$, and $FN$ denote, respectively, the numbers of true positives, false positives, and false negatives as related to detection of cars (the selected relevant class).

## 5.5 Experiments

In this section, we present experimental results derived from applying the AMDO framework on the vehicle classification application introduced in Section 5.4.

### 5.5.1 FSM Parameterization

Recall from Section 6.3 that an AMDO system can be expressed by a tuple $(S_{DHMM}, P, T)$, where the elements of this tuple specify the state machine, parameterization, and PAA set for the system. In our experimental vehicle classification

system, the employed $S_{DHMM}$ is illustrated in Figure 5.2. The PAA set consists of 3 actors, which provide performance assessment in terms of deadline miss rate, execution speed, and remaining battery capacity.

The FSM parameterization $P$ that we employed in our experiments can be expressed as $P = (p_1, p_2, p_3, p_4, p_5)$. Here, $p_1$ represents the deadline for processing each image (i.e., the reciprocal of the supported image processing throughput); $p_2$ represents the *deadline miss tolerance*, which specifies what percentage of deadlines can be missed before the system is considered to be underperforming in terms of real-time operation; $p_3$ represents an analogous tolerance on execution-time over-performance — the system is considered to be overperforming in terms of execution time if the average execution time of an application mode is less than the product $(p_1 \times p_3)$; $p_4$ specifies what percentage of system battery capacity must be exceeded for the system to be overperforming in terms of energy availability; and similarly, $p_5$ specifies a minimum threshold (percentage) on battery capacity below which the system is considered to be underperforming in terms of battery capacity. Collectively, the five parameters in the vector $P$ defined above control how the PAAs in $S_{DHMM}$ cooperate, in a deeply data-driven manner, to explore different regions of the overall design evaluation space — as these parameters are varied, different design trade-offs are concretely realized.

This particular parameterization $P$ is one specific parameterization that we experimented with to concretely demonstrate the AMDO framework; other parameterizations can be derived to drive data-driven, multiobjective optimization in different ways. A central contribution of the AMDO framework is to structure and

raise the level of abstraction in data-driven multiobjective optimization by introducing this kind of parameterization as a first class citizen in the design process for ASM systems. This is an advance over conventional methods for ASM system implementation, which focus on ad-hoc fine-tuning of control code, on analysis of static (non-data-driven) design configurations, or on individual design metrics in isolation.

Our vehicle classification case study, illustrated in Figure 5.3, involved a three step process.

### 5.5.2 Experimental Setup

Our vehicle classification case study, illustrated in Figure 5.3, involved a three step process.

1. We implemented the classifier subsystems in MATLAB for offline training and for generating required classifier parameters (i.e., bias, and support vectors). These off-line-generated parameters were then applied for real-time classification tasks on the targeted mobile platform (described below). The MATLAB implementation was also used for functional validation.

2. Using the LiD4E environment described in Section 5.1, we have also implemented the classifier subsystems (application modes) employed for vehicle classification on a mobile platform (Android-based, Nexus 7, first-generation tablet). We performed extensive profiling of the performance of these mobile-device-targeted subsystem implementations. Data from this mobile-device-

based profiling, including execution time and energy consumption data, was employed to provide characterizations of classifier operation that were applied in the simulation model.

3. We have implemented a simulation model for the vehicle classification system on a desktop computer using the model-based design approach underlying AMDO. The developed simulation environment provides validation of the vehicle classification functionality, along with multidimensional performance assessment of system operation.

Figure 5.4 through Figure 5.6 show execution-time-related profiling data extracted from running the application setup on the targeted mobile platform.

We used 561 vehicle silhouettes from the Statlog dataset [62] for training and 281 images for experimentation. The image sets for training and testing were chosen randomly. We made a minor modification to the annotations in the Statlog dataset by combining the two distinct class labels for cars, "Saab" and "Opel", into a single class labeled "cars". Hence, as described in Section 5.1, our modified dataset consists of three class labels in total — buses, cars, and vans.

### 5.5.3 Experimental Results

Using the AMDO system design and experimental setup described in Section 5.5.1 and Section 5.5.2, we simulated 26 different design points corresponding to 26 different configurations of the FSM parameter SET $P$. The alternative combinations of parameter settings were selected manually with a view towards exper-

imenting with diverse combinations of parameter settings. Alternatively, one could generate and simulate parameter settings using an automated approach, such as an approach that employs a multiobjective evolutionary algorithm (e.g., see [63]) to maintain populations of parameter settings, and employs our AMDO simulation framework for fitness evaluation. Such automated design space exploration using the AMDO framework is a useful direction for future work.

As discussed in Section 5.4, the design evaluation metrics considered in our experiments are throughput; deadline miss rate; energy efficiency; and both precision and recall for detecting cars. Here, energy efficiency is measured as the number of images that were processed (excluding deadline misses) for a given amount of initial battery capacity. The amount of initial battery capacity employed in the experiments was 432.5 milliampere-hours (mAh). The metric employed for energy efficiency thus gives an indication of the total volume of data that can be processed before the given amount of battery capacity expires.

Table 5.1 lists the set of Pareto-optimal designs from among the set $Y$ of 26 design points that we generated in our experiments. Here, we say that a point $y \in Y$ is Pareto-optimal if for any other point $y\prime \in Y$, $y\prime$ is inferior to $y$ in terms of at least one design evaluation metric. For general background on Pareto optimization in the context of electronic system design, we refer the reader to [64]. Intuitively, a Pareto point represents a useful design point to keep track of during design space exploration because such a design point cannot be improved upon in any dimension without sacrificing quality in at least one other dimension. Among the 26 design points explored in our experiments, 16 (61%) were found to be Pareto-optimal.

These 16 points are the ones that are listed in Table 5.1 along with their simulated performance results in terms of the five targeted design evaluation metrics.

In summary, the experiments and results presented in this section demonstrate concretely how AMDO enables designers to rapidly investigate diverse sets of alternative design points for an ASM system (1) relative to a complex multidimensional design evaluation space, and (2) in a manner that systematically takes into account data-driven adaptation of application modes and system implementation parameters within a unified framework.

## 5.6 Summary

In this chapter, we have introduced a new multiobjective design optimization framework for adaptive stream stream mining systems (ASMs). The framework, called the ASM multiobjective design optimization (AMDO) framework, employs a novel design methodology centered on data-driven control of and coordination between alternative dataflow subsystems for stream mining. AMDO allows system designers to efficiently explore complex, multidimensional design evaluation spaces in a data-driven manner, and is readily adaptable to different kinds of operational constraints and objectives. We have integrated AMDO into the Lightweight Dataflow for DDDAS Environment (LiD4E) tool for design and implementation ASM systems, and demonstrated the framework using a case study involving real-time and energy-constrained multiclass vehicle classification. Useful directions for future work include development of automated design space exploration methods

| Design ID | Energy Efficiency (images processed) | Deadline Miss Rate (%) | Throughput (images per second) | Average Precision for Cars (%) | Average Recall for Cars (%) |
|---|---|---|---|---|---|
| AMDO-1 | 861237 | 0.13 | 83.33 | 99.15 | 95.42 |
| AMDO-2 | 847853 | 0.36 | 90.91 | 99.15 | 95.49 |
| AMDO-3 | 679407 | 2.01 | 90.91 | 98.27 | 97.34 |
| AMDO-4 | 656775 | 0.36 | 66.67 | 97.90 | 97.90 |
| AMDO-5 | 861438 | 0.07 | 66.67 | 99.15 | 95.42 |
| AMDO-6 | 651649 | 4.20 | 100.00 | 98.09 | 97.61 |
| AMDO-7 | 656566 | 0.35 | 62.50 | 97.90 | 97.90 |
| AMDO-8 | 821935 | 0.06 | 62.50 | 99.02 | 96.76 |
| AMDO-9 | 861654 | 0.03 | 62.50 | 99.15 | 95.42 |
| AMDO-10 | 861312 | 0.10 | 76.92 | 99.15 | 95.42 |
| AMDO-11 | 861162 | 0.13 | 83.33 | 95.42 | 99.15 |
| AMDO-12 | 653875 | 1.06 | 83.33 | 97.90 | 97.90 |
| AMDO-13 | 849897 | 0.03 | 62.50 | 99.15 | 95.50 |
| AMDO-14 | 723423 | 18.13 | 135.14 | 99.27 | 95.17 |
| AMDO-15 | 18470 | 98.09 | 169.49 | 99.27 | 95.10 |
| AMDO-16 | 155060 | 83.75 | 166.67 | 99.27 | 95.10 |

Table 5.1: Pareto-optimal design points derived through design space exploration.

using the AMDO framework, such as integration of AMDO methods with multiobjective evolutionary algorithms.

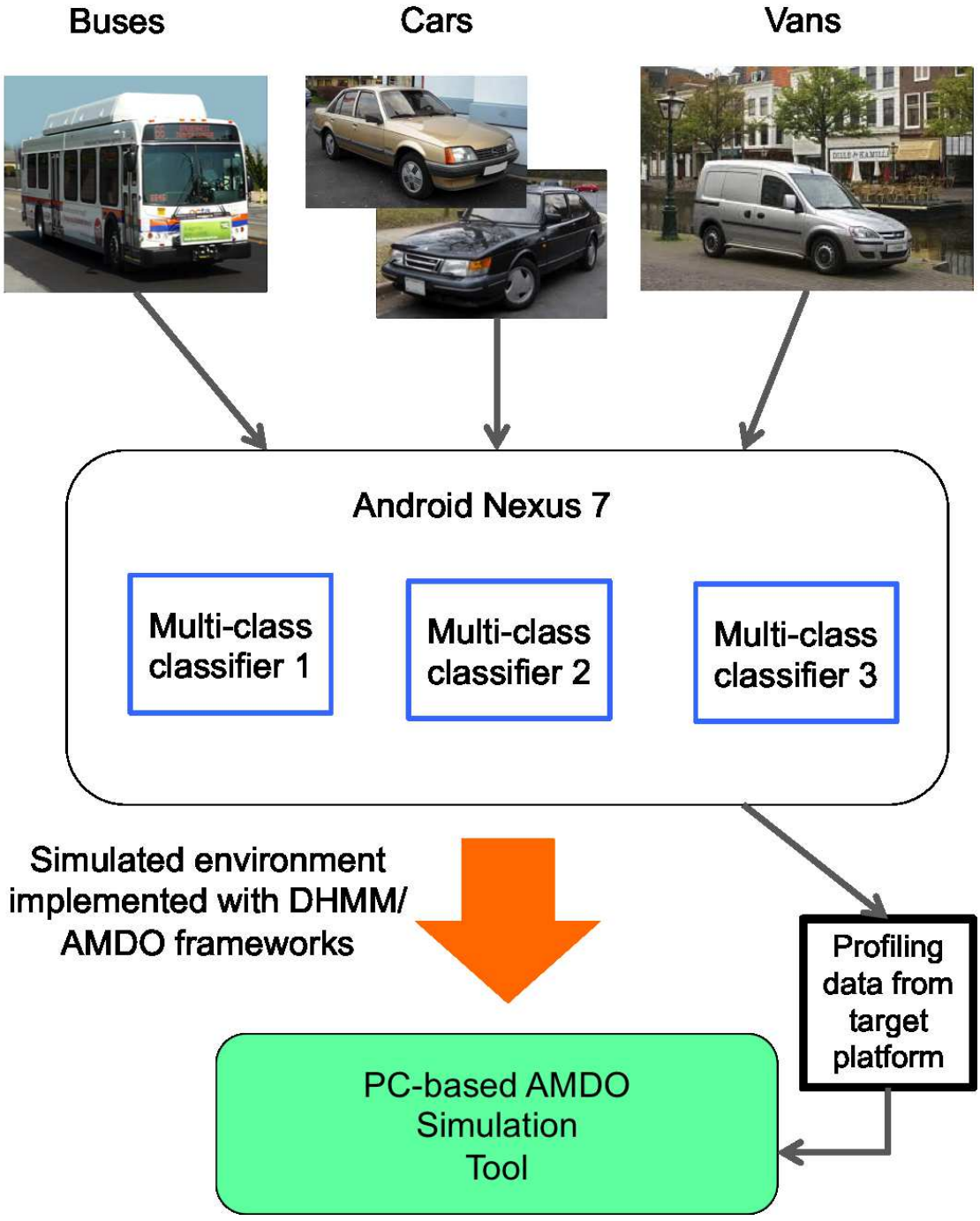


Figure 5.3: An illustration of the vehicle classification system that is employed in our experiments.

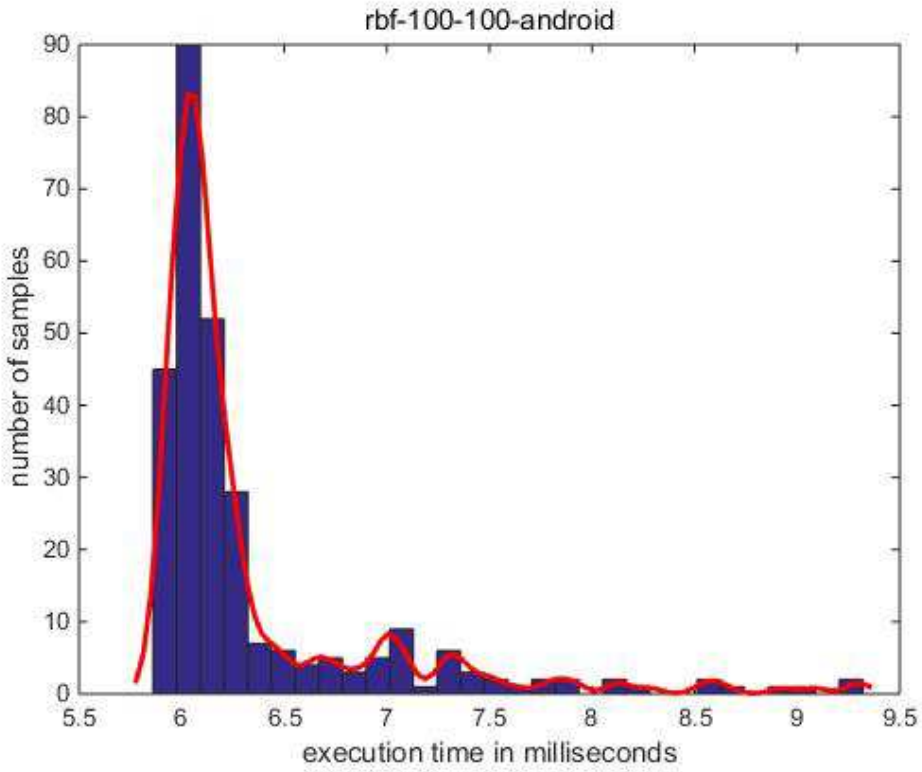


Figure 5.4: Task execution time for classifier subsystem 1.

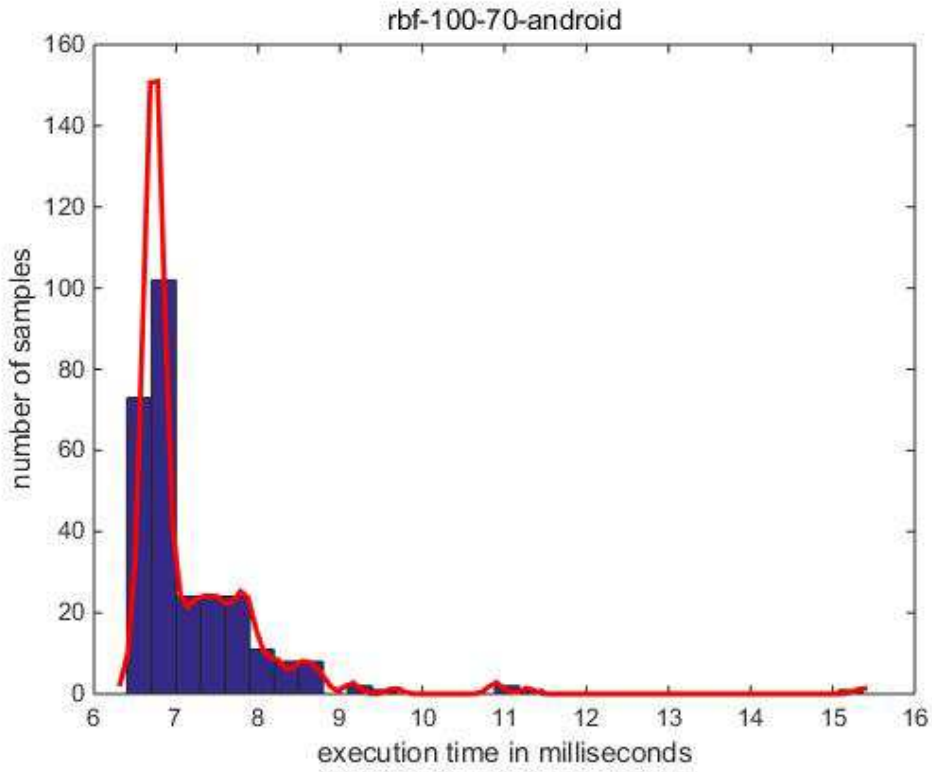


Figure 5.5: Task execution time for classifier subsystem 2.

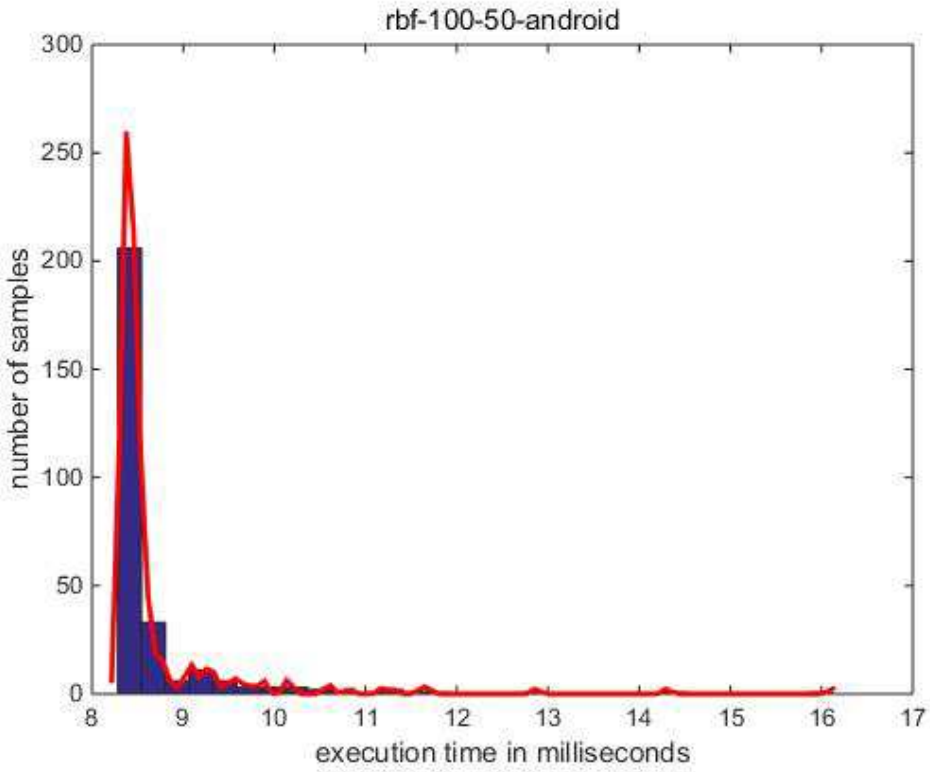


Figure 5.6: Task execution time for classifier subsystem 3.

# Chapter 6: Conclusion and Future Work

In this thesis, we have motivated the need for new design methods for the emerging domain of adaptive stream mining systems, which encompasses a growing class of applications at the intersection of big data processing, ubiquitous sensing, and real-time systems. We have also motivated how the paradigm of Dynamic Data Driven Applications Systems (DDDAS), and the methodology of dataflow-based design of signal processing systems offer valuable concepts and methods that are relevant to the adaptive stream mining domain. This thesis represents an effort to integrate and apply the methods of DDDAS and dataflow to develop new design methods and associated software tools that address implementation challenges for adaptive stream mining systems.

The contributions of this thesis can be categorized into three main areas:

- Hierarchical dataflow modeling and supporting design tools for dynamic, data-driven signal processing systems;
- design and scheduling for multi-mode stream mining; and
- multiobjective design optimization in the context of the aforementioned areas.

In the remainder of this chapter, we summarize our contributions in these

areas, and outline useful directions for future work.

## 6.1 Hierarchical Dataflow Modeling

In chapter 3, we introduced the hierarchical core functional dataflow (HCFDF) model of computation, and demonstrated its utility in modeling adaptive stream mining systems. HCFDF is derived by integrating hierarchical semantics into the previously-developed core functional dataflow (CFDF) model [11, 12]. HCFDF is designed to help manage complexity in dynamic dataflow designs, while enforcing consistency with CFDF semantics throughout the design hierarchy.

To facilitate design and implementation of adaptive stream mining systems using dataflow methods and the hierarchical design capabilities of HCFDF, we developed a novel design tool called lightweight dataflow for dynamic data driven application systems (LiD4E). LiD4E extends the lightweight dataflow environment (LIDE) [13,14] with new capabilities for modeling, implementation, and design space exploration, as well as capabilities for applying the DDDAS paradigm efficiently to manage system configurations for stream mining and data-driven signal processing.

We demonstrated the capabilities of LiD4E using a face detection application in which alternative classifiers with different trade-offs are available at run-time, and the applied classifier is dynamically adapted to strategically align system operation with time varying application constraints. Through this case study, we demonstrated the ability of LiD4E and the underlying HCFDF modeling techniques to provide systematic, reliable, and, efficient adaptation of system performance.

Useful directions for future work on LiD4E include the integration of parametric dataflow techniques, such as those provided by the parameterized dataflow and parameterized and interfaced dataflow (PiMM) meta-models [65, 66]. Such techniques provide the potential for efficient quasi-static scheduling through parameterized scheduling structures. The investigation of such quasi-static scheduling techniques in connection with parametric dataflow models is an interesting research direction that has potential for further improvements in efficiency, modeling flexibility, and reliability through formal guarantees of system properties.

## 6.2 Multi-mode System Design

Building on the modeling methods and tools described in chapter 3, we addressed the design and implementation of multi-mode stream mining systems in Chapter 4. In this chapter, we introduced the DHMM (DDDAS-HCFDF-Multi-Mode) scheduling framework for modeling and management of alternative stream mining application modes. An important aspect of this scheduling framework and the associated DHMM design methodology is the modeling of DDDAS-oriented instrumentation systems as first class citizens in the design process. Thus, in the DHMM approach, dataflow methods provide a unified approach for specifying and integrating application behavior, decomposed as alternative operational modes; instrumentation behavior; and reconfiguration behavior that adapts future modes and instrumentation operations based on the analysis of data from recent instrumentation measurements.

A key component of the DHMM design methodology is the use of a state machine, called the DHMM state machine, as a standard subsystem for coordinating DDDAS-based integration of instrumentation, application execution, and configuration management throughout the stream mining process. Each state in the DHMM state machine encapsulates a function for configuring system parameters based on the application mode represented by that state, while transitions between states are triggered when analysis of instrumentation data indicates that migration to a different mode would be better aligned in terms of current data characteristics and system requirements. The DHMM design methodology centers on the integrated use of (1) such state machine driven coordination across different application modes, along with (2) the HCDF-based modeling of individual modes and the instrumentation functionality that is used to guide state transitions and intra-state configuration functionality.

## 6.3 Multiobjective Design Optimization

In Chapter 5, we augmented the DHMM scheduling framework developed in Chapter 4 with new capabilities for modeling, navigation, and optimization involving complex, multidimensional design evaluation spaces. This work is motivated by the need to consider adaptive configuration of system operation across diverse, often-competing implementation metrics, including power and energy consumption, real-time performance, and multidimensional metrics that are commonly associated with stream mining quality of service (e.g., detection accuracy and false positive rates).

To address this challenge, we introduced in Chapter 5 the Adaptive stream mining Multiobjective Design Optimization (AMDO) framework. The AMDO framework inherits from DHMM the HCDF modeling approach, and the state-machine-based coordination involving instrumentation and configuration management. Additionally, AMDO introduces methods for parameterizing the DHMM state machine and for applying state machine parameters in a systematic fashion to periodically assess system performance (based on the underlying DDDAS-based instrumentation subsystem), and drive state transition logic based on such assessments. The AMDO framework provides a novel approach for parametrically managing how data driven adaption is controlled, and how multidimensional design evaluation metrics are consequently traded off throughout the execution process.

The proposed AMDO framework is developed as a methodology in which design and implementation of stream mining systems is structured through the integrated design of DHMM state machines, state machine parameterizations, application subsystems, instrumentation subsystems, and specialized dataflow components called performance assessment actors (PAAs). Useful directions for future work include development of automated design space exploration methods using the AMDO framework, such as integration of AMDO methods with multiobjective evolutionary algorithms [63, 67, 68].

# Bibliography


[1] L. Amini, H. Andrade, F. Eskesen, R. King, Y. Park, P. Selo, and C. Venkatramani. The stream processing core. Technical Report RSC 23798, 2005.

[2] R. Ducasse, D. Turaga, and M. van der Schaar. Adaptive topologic optimization for large-scale stream mining. *IEEE Journal on Selected Topics in Signal Processing*, 4(3):620–636, June 2010.

[3] C. Olston, J. Jiang, and J. Widom. Adaptive filters for continuous queries over distributed data streams. In *ACM SIGMOD*, 2003.

[4] E. A. Lee and T. M. Parks. Dataflow process networks. *Proceedings of the IEEE*, pages 773–799, May 1995.

[5] S. S. Bhattacharyya, E. Deprettere, R. Leupers, and J. Takala, editors. *Handbook of Signal Processing Systems*. Springer, second edition, 2013. ISBN: 978-1-4614-6858-5 (Print); 978-1-4614-6859-2 (Online).

[6] F. Darema. Dynamic data driven applications systems: A new paradigm for application simulations and measurements. In *Proceedings of the International Conference on Computational Science*, pages 662–669, 2004.

[7] F. Darema. Grid computing and beyond: The context of dynamic data driven applications systems. *Proceedings of the IEEE*, 93(2):692–697, 2005.

[8] E. A. Lee and D. G. Messerschmitt. Synchronous dataflow. *Proceedings of the IEEE*, 75(9):1235–1245, September 1987.

[9] G. Bilsen, M. Engels, R. Lauwereins, and J. A. Peperstraete. Cyclo-static data flow. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, pages 3255–3258, May 1995.

[10] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee. *Software Synthesis from Dataflow Graphs*. Kluwer Academic Publishers, 1996.

[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] W. Plishker, N. Sane, M. Kiemb, and S. S. Bhattacharyya. Heterogeneous design in functional DIF. In *Proceedings of the International Workshop on Systems, Architectures, Modeling, and Simulation*, 2008.

[13] C. Shen, W. Plishker, H. Wu, and S. S. Bhattacharyya. A lightweight dataflow approach for design and implementation of SDR systems. In *Proceedings of the Wireless Innovation Conference and Product Exposition*, pages 640–645, Washington DC, USA, November 2010.

[14] C. Shen, W. Plishker, and S. S. Bhattacharyya. Dataflow-based design and implementation of image processing applications. In L. Guan, Y. He, and S.-Y. Kung, editors, *Multimedia Image and Video Processing*, pages 609–629. CRC Press, second edition, 2012. Chapter 24.

[15] R. Ducasse and M. van der Schaar. Finding it now: Construction and configuration of networked classifiers in real-time stream mining systems. In S. S. Bhattacharyya, E. F. Deprettere, R. Leupers, and J. Takala, editors, *Handbook of Signal Processing Systems*, pages 97–134. Springer, second edition, 2013.

[16] S. Sriram and S. S. Bhattacharyya. *Embedded Multiprocessors: Scheduling and Synchronization*. CRC Press, second edition, 2009. ISBN:1420048015.

[17] 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.

[18] E. A. Lee and D. G. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Transactions on Computers*, 36(1):24–35, 1987.

[19] H. Topcuoglu, S. Hariri, and M.-Y. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):260–274, 2002.

[20] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier. StarPU: a unified platform for task scheduling on heterogeneous multicore architectures. *Journal of Concurrency and Computation: Practice & Experience*, 23(2):187–198, February 2011.

[21] J. T. Buck. *Scheduling Dynamic Dataflow Graphs with Bounded Memory using the Token Flow Model*. PhD thesis, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, September 1993.

[22] W. Plishker, N. Sane, M. Kiemb, and S. S. Bhattacharyya. Heterogeneous design in functional DIF. In P. Stenström, editor, *Transactions on High-Performance Embedded Architectures and Compilers IV*, volume 6760 of *Lecture Notes in Computer Science*, pages 391–408. Springer Berlin / Heidelberg, 2011.

[23] S. Ritz, M. Pankert, and H. Meyr. High level software synthesis for signal processing systems. In *Proceedings of the International Conference on Application Specific Array Processors*, August 1992.

[24] S. Ritz, M. Pankert, and H. Meyr. Optimum vectorization of scalable synchronous dataflow graphs. In *Proceedings of the International Conference on Application Specific Array Processors*, October 1993.

[25] M. Ko, C. Shen, and S. S. Bhattacharyya. Memory-constrained block processing for DSP software optimization. *Journal of Signal Processing Systems*, 50(2):163–177, February 2008.

[26] S. S. Bhattacharyya, M. van der Schaar, O. Atan, C. Tekin, and K. Sudusinghe. Data-driven stream mining systems for computer vision. In B. Kisacanin and M. Gelautz, editors, *Advances in Embedded Computer Vision*, Advances in Computer Vision and Pattern Recognition, pages 249–264. Springer, 2014.

[27] B. Plale, D. Gannon, D. Reed, S. Graves, K. Droegemeier, B. Wilhelmson, and M. Ramamurthy. Towards dynamically adaptive weather analysis and forecasting in LEAD. In *Proceedings of the International Conference on Computational Science*, pages 624–631, 2005.

[28] A. Majumdar, A. Birnbaum, D. J. Choi, A. Trivedi, S. K. Warfield, K. Baldridge, and P. Krysl. A dynamic data driven grid system for intra-operative image guided neurosurgery. In *Proceedings of the International Conference on Computational Science*, pages 672–679, 2005.

[29] D. Metaxas and G. Tsechpenakis. Dynamic data driven coupling of continuous and discrete methods for 3D tracking. In *Proceedings of the International Conference on Computational Science*, pages 712–720, 2005.

[30] M. A. Hearst, S. T. Dumais, E. Osman, J. Platt, and B. Scholkopf. Support vector machines. *IEEE Intelligent Systems and their Applications*, 13(4):18–28, 1998.

[31] V. D. Sánchez. Advanced support vector machines and kernel methods. *Neurocomputing*, 55:5–20, 2003.

[32] C. Campbell. Kernel methods: a survey of current techniques. *Neurocomputing*, 48:63–84, 2002.

[33] I. Chukhman and S. S. Bhattacharyya. Instrumentation-driven framework for validation of dataflow applications. In *Proceedings of the IEEE Workshop on Signal Processing Systems*, pages 1–6, Belfast, UK, October 2014.

[34] K. Sudusinghe, S. Won, M. van der Schaar, and S. S. Bhattacharyya. A novel framework for design and implementation of adaptive stream mining systems. In *Proceedings of the IEEE International Conference on Multimedia and Expo*, pages 1–6, San Jose, California, July 2013.

[35] S. Chandrasekaran, O. Cooper, A. Deshpande, M. Franklin, J. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. Shah. Telegraphcq: Continuous dataflow processing for an uncertain world. In *CIDR*, 2003.

[36] 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*, 2011.

[37] B. D. Theelen, M. C. W. Geilen, T. Basten, J. P. M. Voeten, S. V. Gheorghita, and S. Stuijk. A scenario-aware data flow model for combined long-run average and worst-case performance analysis. In *Proceedings of the International Conference on Formal Methods and Models for Codesign*, July 2006.

[38] J. Piat, S. S. Bhattacharyya, and M. Raulet. Loop transformations for interface-based hierarchies in SDF graphs. In *Proceedings of the International Conference on Application Specific Systems, Architectures, and Processors*, 2010.

[39] A. Girault, B. Lee, and E. A. Lee. Hierarchical finite state machines with multiple concurrency models. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(6):742–760, June 1999.

[40] C. Shen, W. Plishker, H. Wu, and S. S. Bhattacharyya. A lightweight dataflow approach for design and implementation of SDR systems. In *Proceedings of the Wireless Innovation Conference and Product Exposition*, 2010.

[41] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Knowledge Discovery and Data Mining*, 2(2), 1998.

[42] B. Heisele, P. Ho, J. Wu, and T. Poggio. Face recognition: component-based versus global approaches. *Journal of Computer Vision and Image Understanding*, 91(1–2):6–21, 2003.

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

[44] K. Sudusinghe, I. Cho, M. van der Schaar, and S. S. Bhattacharyya. Model based design environment for data-driven embedded signal processing systems. In *Proceedings of the International Conference on Computational Science*, pages 1193–1202, Cairns, Australia, June 2014.

[45] W. Najjar, B. Draper, W. Bohm, and R. Beveridge. The cameron project: High-level programming of image processing applications on reconfigurable computing machines. In *Proceedings of the PACT Workshop on Reconfigurable Computing*, 1998.

[46] J. B. Dennis. Dataflow supercomputers. *Computer*, 13(11), November 1980.

[47] U. Ramacher. Software-defined radio prospects for multistandard mobile phones. *Computer*, 40(10):62–69, 2007.

[48] J. Pisharath, N. Jiang, and A. Choudhary. Evaluation of application-aware heterogeneous embedded systems for performance and energy consumption. In *Proceedings of the IEEE Real-Time Technology and Applications Symposium*, pages 124–132, 2003.

[49] F. König, D. Boers, F. Slomka, U. Margull, M. Niemetz, and G. Wirrer. Application specific performance indicators for quantitative evaluation of the timing behavior for embedded real-time systems. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, pages 519–523, 2009.

[50] G. Chollet, K. McTait, and D. Petrovska-Delacrétaz. Data driven approaches to speech and language processing. In *Nonlinear Speech Modeling and Applications*, pages 164–198. Springer, 2005.

[51] G. Aradilla, J. Vepa, and H. Bourlard. Improving speech recognition using a data-driven approach. Technical Report IDIAP-RR 05-66, IDIAP Research Institute, April 2005.

[52] H. Ney, D. Mergel, A. Noll, and A. Paeseler and. Data driven search organization for continuous speech recognition. *IEEE Transactions on Signal Processing*, 40(2):272–281, 1992.

[53] G. Bilsen, M. Engels, R. Lauwereins, and J. A. Peperstraete. Cyclo-static dataflow. *IEEE Transactions on Signal Processing*, 44(2):397–408, February 1996.

[54] C. Shen, W. L. Plishker, D. Ko, S. S. Bhattacharyya, and N. Goldsman. Energy-driven distribution of signal processing applications across wireless sensor networks. *ACM Transactions on Sensor Networks*, 6(3), June 2010. Article No. 24, 32 pages, DOI:10.1145/1754414.1754420.

[55] I. Estevez-Ayres, M. Garcia-Valls, and P. Basanta-Val. Enabling WCET-based composition of service-based real-time applications. *ACM SIGBED Review*, 2(3):25–29, 2005.

[56] J. Fredriksson. Increasing accuracy of property predictions for embedded real-time components. In *Proceedings of the Euromicro Conference on Real-Time Systems*, 2006.

[57] R. Wilhelm et al. The worst-case execution-time problem — overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems*, 7(3), 2008.

[58] R. Perrone, R. Macedo, G. Lima, and V. Lima. An approach for estimating execution time probability distributions of component-based real-time systems. 15(11):2142–2165, 2009.

[59] K. Sudusinghe, Y. Jiao, H. Ben Salem, M. van der Schaar, and S. S. Bhattacharyya. Multiobjective design optimization in the lightweight dataflow for DDDAS environment (LiD4E). In *Proceedings of the International Conference on Computational Science*, Reykjavik, Iceland, June 2015. To appear.

[60] C.-W. Hsu and C.-J. Lin. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425, 2002.

[61] A. Yilmaz, O. Javed, and M. Shah. Object tracking: A survey. *ACM Computing Surveys*, 38(4), 2006.

[62] K. Bache and M. Lichman. UCI machine learning repository, 2013.

[63] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, November 1999.

[64] G. De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, 1994.

[65] K. Desnos, M. Pelcat, J.-F. Nezan, S. S. Bhattacharyya, and S. Aridhi. PiMM: Parameterized and interfaced dataflow meta-model for MPSoCs runtime reconfiguration. In *Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, pages 41–48, Samos, Greece, July 2013.

[66] B. Bhattacharya and S. S. Bhattacharyya. Parameterized dataflow modeling for DSP systems. *IEEE Transactions on Signal Processing*, 49(10):2408–2421, October 2001. DOI:10.1109/78.950795.

[67] A. Mukhopadhyay, U. Maulik, S. Bandyopadhyay, and C. A. C. Coello. A survey of multiobjective evolutionary algorithms for data mining: Part i. *IEEE Transactions on Evolutionary Computation*, 18(1):4–19, 2014.

[68] A. Mukhopadhyay, U. Maulik, S. Bandyopadhyay, and C. A. C. Coello. A survey of multiobjective evolutionary algorithms for data mining: Part ii. *IEEE Transactions on Evolutionary Computation*, 18(1):20–35, 2014.