Data-Driven Cyber-Physical Systems via Real-Time Stream Analytics and Machine Learning

by

Ilge Akkaya

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering - Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Edward A. Lee, Chair
Professor Sanjit A. Seshia
Professor Anil J. Aswani

Fall 2016

Data-Driven Cyber-Physical Systems via Real-Time Stream Analytics and Machine Learning

Abstract

Data-Driven Cyber-Physical Systems via Real-Time Stream Analytics and Machine Learning

by

Ilge Akkaya
Doctor of Philosophy in Engineering - Electrical Engineering and Computer Sciences

University of California, Berkeley
Professor Edward A. Lee, Chair

Emerging distributed cyber-physical systems (CPSs) integrate a wide range of heterogeneous components that need to be orchestrated in a dynamic environment. While model-based techniques are commonly used in CPS design, they become inadequate in capturing the complexity as systems become larger and extremely dynamic. The adaptive nature of the systems makes data-driven approaches highly desirable, if not necessary.

Traditionally, data-driven systems utilize large volumes of static data sets to extract models and predictions of physical processes. However, in emerging CPS, networked sensors provide continually streaming data, creating an essentially *infinite* source of information. Processing data in batches is no longer a viable option: streams are most valuable when processed on-line, allowing actionable information to be gathered just as the data becomes available. This fundamental shift from *big data* to *infinite data*, while having great potential to enable smarter systems, also poses unique challenges. Computation models that capture the integration of streaming data into CPS design become a key requirement for systems to learn, adapt, and evolve in real-time.

This thesis explores methodologies for developing data-driven CPSs that integrate model-based design and real-time stream analytics in a modular way. The key modeling framework to be introduced is the aspect-oriented modeling (AOM) paradigm, which leverages the principle of separation-of-concerns in actor-oriented design. *Aspects* are useful for representing cross-cutting concerns in complex system architectures, as first introduced by the aspect-oriented programming paradigm in object-oriented design. AOM applies this idea to actor-oriented design, creating *aspects* that enable representation of modular concerns in a complex system model. In data-driven CPS, the introduced aspects can be leveraged to process streaming data, extract actionable information, and incorporate these into the system workflow in a way that preserves model semantics and modularity. To address information extraction from streaming data, we propose the use of aspects that implement Dynamic

Bayesian Network based algorithms for machine learning and optimization. Specifically, we introduce an actor-oriented toolkit that enables dynamics and sensing models to be composed with inference, Bayesian learning, and optimization algorithms, and present comprehensive case studies on cooperative mobile robot control. We additionally study the use of streaming data for control of dynamic networked CPS in the context of home automation, and present an overview of the use cases of aspects in actor-oriented CPS development.

To my parents, grandparents, Idil, and Mert.

Contents

List of Figures

List of Tables

Acknowledgments

First and foremost, I would like to thank my PhD advisor Prof. Edward A. Lee for his continued support and mentorship throughout my studies. Under his guidance, I had the opportunity to collaborate with a number of great researchers and explore many interesting research directions. His ideas and feedback were nothing short of inspiring and I always felt encouraged to explore connections between diverse topics, which I greatly enjoyed. I would also like to sincerely thank Professors Sanjit Seshia, Anil Aswani, and David Wessel for serving on my qualification exam and dissertation committees. Their feedback and continued guidance greatly helped shape my research and dissertation.

During the wonderful years I spent at Berkeley, I had the opportunity to collaborate with a number of excellent researchers. I consider myself very fortunate to have collaborated with Prof. Sanjit Seshia and his research group, many members of CNMAT, as well as with Shuhei Emoto and other visitors from IHI. Furthermore, I would like to thank a number of people I had the opportunity to co-author with: Yan Liu, Marjan Sirani, John Eidson, Patricia Derler, Alexandre Donzé, Rafael Valle, and Daniel Fremont.

Next, I would like to acknowledge all the past and present members of the Ptolemy group and the DOP center: Ben Zhang, Marten Lohstroh, Fabio Cremona, Hokeun Kim, Chris Shaver, Matt Weber, Antonio Iannopollo, Pierluigi Nuzzo, and many others, for the valuable discussions and their support in general. Also, my special thanks go to Christopher Brooks, Mary Stewart, and Shirley Salanio for always being available and helpful.

I would like to acknowledge my friends and family for being extremely supportive throughout my studies. I consider myself lucky to have such amazing parents and grandparents who supported me in every way and raised me with the courage to take the less-traveled roads to pursue my interests. Last but not least, I would like to thank my fiancé Mert for his love and support. Without the help and vision of these people, I would never have made the decisions I have made.

Life indeed is a random walk, and I am so happy to have spent the past six years at Cal and have met the people I have, who have made it extremely worthwhile.

Chapter 1

Introduction

Cyber-Physical Systems (CPSs) is a term that defines an orchestration of physical components and computational processes, where 'cyber' and 'physical' parts are tightly coupled in a feedback relation, continuously affecting one another. This property of CPSs renders the separate understanding of the physical and the cyber insufficient, and requires CPS research to focus on the *intersection* of the two rather than the union by developing formalisms that require a complete understanding of the cyber-physical interaction [83, 78]. Advancements in this domain have recently enabled disruptive applications, most notably in industrial automation, manufacturing, transportation, wearable technologies, and energy systems. Multiple research directions have emerged to realize systems sharing the theme of *connecting billions of devices to humans and their environment*, realizing large-scale "smart" systems. One such direction that has been receiving significant attention from researchers and investors is the Internet-of-Things (IoT).

IoT systems, by definition, are a realization of CPSs, where the concept of intensively networked components is highly emphasized. IoT envisions leveraging Internet technology to connect and unprecedented number of devices, yielding a "swarm" of heterogeneous sensors and actuators that can interact with the physical environment and can be used to enable dynamic decision making in many different domains [79, 80]. Of course, the resulting device swarms are envisioned to be "smart," continuously learning from behavioral patterns of humans and other devices, then autonomously adapting to changes at run time. This ability builds upon the implicit assumption that systems will be able to make *real-time decisions* on *streaming data*. While the meaningful interpretation of this specification is exceedingly context-dependent, for the purposes of the IoT, the implied requirement is the ability of a system to continually process streams of data in order to extract *actionable information*.

This concept fundamentally differs from traditional computation models, where actions (as outputs of computation) are either based on a priori information alone, therefore require no streaming data, or on batch data collected over time, that can be processed as a whole and in an off-line manner. The fundamental shift in the information source has promoted novel approaches in retrieval, storage, and processing of data, all of which can be summarized under the term *stream analytics*. Forrester, Inc. has coined the term *perishable insights* to refer to information that must be acted upon fast, as insights obtained from streaming data (such as market data, sensors, clickstream, and transactions) quickly lose their value if they were to be processed in 'batch mode' [49]. The concept of perishable insights has a much more intuitive interpretation for cyber-physical systems. In a CPS, data streams are most beneficial at the time they are produced, as any change reported by the data (e.g. a sensor anomaly, a fault in the physical process being sensed, or a change of system state) should be detected as soon as possible, and be acted upon, for example, via a change in control policy or an output action. Furthermore, as opposed to stream analytics for purely software systems, in CPS, the insight being revealed by data will often be tied to a safety-critical action that must be performed to ensure the health of the CPS itself.

As an example, consider the electric power grid, which is an extremely large-scale distributed cyber-physical energy system spanning hundreds of generators, loads, and regulatory units that are interconnected with distributed computation and monitoring nodes. Traditionally, the health of this complex CPS has relied on local circuits at generators as well as state estimation algorithms that operate on extremely low fidelity data delivered by the supervisory control and data acquisition (SCADA) system [22]. The physical components making up grids have to operate in close synchrony to avoid physical damage to loads and generators. The North American grid, for example, operates at 60 Hz nominally with a ± 0.5 Hz daily oscillation range. Figure 1.1 depicts a snapshot of the real-time grid frequency over the North American Interconnections (Western, Texas, Eastern, and Quebec regions), that are frequency-independent islands operating in precise synchrony within themselves, connected via load-balancers that ensure healthy transfer between interconnected areas. Note that each interconnection has to maintain a common local frequency with a maximum frequency deviation of only several mHz [6]. This real-time mapping technology is made possible only recently, with the addition of high-throughput time-synchronized sensors — known as *synchrophasors* — installed at thousands of grid nodes [87]. Real-time snapshots of grid frequency and many more types of streaming data are made available. One example obtained by the web interface of the FNET/GridEye synchrophasor deployment is presented in Figure 1.1. An out-of-phase generator in the grid is easy to detect (and also predict) given a real-time map of grid frequency as this one, realized by stream processing of synchrophasor data. Using streaming data in batch mode in many scenarios including this one would not go beyond postmortem analysis of a catastrophic system-wide failure [34].

Naturally, emerging IoT applications typically require much more complex computations on data streams, including analyzing multiple seemingly unrelated data streams via statistical inference to estimate state or to detect events, when direct observation is not possible.

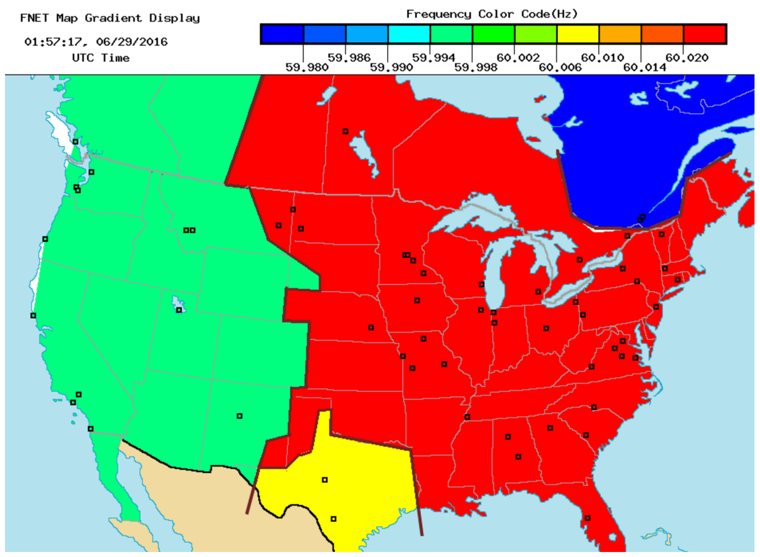


Figure 1.1: FNET/GridEye real-time frequency gradient map

This paradigm shift is becoming apparent in the design of modern control systems. In classical control, sensor data provide a direct measurement of a physical quantity to be controlled. For example, an encoder measures the position of a rotating shaft, or an accelerometer measures the tilt of an object to be balanced. In modern control systems, the data provided by sensors is more indirect. Consider, for example, a robot that is trying to track a target. It may use a combination of sensors, including cameras, laser or ultrasonic rangefinders, power measurements of radio signals, microphones, etc. None of these sources present data that directly measures any controllable quantity. First, these data must be converted to actionable information, e.g., a concise summary of the estimated location and trajectory of the target. Then, given the summary, a trade-off between often conflicting objectives must be computed and translated into action. To be effective, the data summarization and optimization must be performed in real time on streaming data.

Although many novel applications have been proposed that aim to use data for such purposes, there remain significant barriers to progress towards the realization of dependable data-driven CPSs that can replace or enhance existing infrastructure. Despite the existence of a plethora of proof of concept implementations of smart cities, smart homes, robot swarms, and many other intelligent systems and infrastructure [71, 47, 55, 74, 24], it remains difficult to build reliable applications that utilize data streams. Traditional design and analysis tools in many domains become unable to cope with the level of complexity required to realize swarm systems at scale. Consequently, it becomes impossible to conceptualize the implications of adaptive applications that operate on streaming data to manipulate the physical world in response [11]. Moreover, despite the envisioned benefits of using continually streaming sensor data for this purpose, concerns about ensuring privacy, security, and meeting adequate data storage requirements quickly overwhelm the potential positive outcomes.

A key hurdle against enabling reliable real-time decision making on streaming data is the lack of new computation models that can handle the new relationships that arise between data streams and physical components. Novel software models that redefine computations on streaming data, tie data-driven insights to actions on continuous system dynamics, verify safety, and evaluate uncertainty on feedback interactions between the 'cyber' and the 'physical' become a necessity. Additionally, the models should address the heterogeneous and often conflicting concerns that govern complex CPS operation, ideally maintaining "separation-of-concerns." Specifically, design patterns should ensure modularity and detangling of implementation details from the functionality being realized, while making it easy to explicitly model how data affects system models and dynamics [80].

A family of approaches that has proven extremely useful in providing this encapsulation for reliable CPS design over decades are known as model-based design (MBD) techniques. MBD focuses on building accurate models of devices and their operating environment, then providing strong guarantees on correctness and safety under given assumptions. MBD is being used intensively in many safety-critical domains including automotive, aerospace, and energy systems [45, 30, 93]. While MBD provides desirable results in provable and scalable development, it often requires a complex set of a priori assumptions about systems, making it difficult to adapt. Obtaining faithful models, as well as adapting models to changing operational conditions are processes that can benefit greatly from operational data. When data collected from system operation becomes relevant to modeling the system-level properties, *data-driven* approaches gain importance over a priori models. Data-driven modeling leverages large volumes of data collected by sensors, devices, web applications, etc., and lead to adaptive probabilistic or deterministic models about the operation of the sensed systems. By using mathematical optimization and machine learning workflows, such data can be utilized to build better system models during system operation. For development of extremely complex CPSs, both model-based, and data-driven techniques will be necessary, especially for those systems expected to operate under uncertain and highly varying conditions, which renders precomputation of control and operation models extremely impractical, and oftentimes impossible.

1.1 Contributions and Overview

This thesis studies an actor-oriented design pattern for streaming data to be effectively utilized in real-time decision making, while ensuring rigorous separation of concerns in CPS design. The goal is to synthesize continually evolving systems that can adapt to environmental changes sensed via streams of data.

The contributions of this work are in introducing the aspect-oriented modeling approach to CPS design and developing a toolkit that presents a collection of inference and machine learning algorithms implemented in the form of aspect-oriented components. The concept of aspects, first introduced for object-oriented design with aspect-oriented programming, aim to provide separation-of-concerns in complex system architectures. Although aspects present significant limitations in object-oriented programs, they are promising when combined with actor-oriented design and used for modular processing of infinite data streams to enable data-driven CPS. The aspect-oriented modeling framework introduced in this thesis enables modeling system-specific concerns such as dynamics, measurement models, noise distributions, and environmental concerns as *aspects*. This abstraction leads to a novel design paradigm that enables inference and learning components that operate on streaming data to share and update information related to the CPS. The novel framework is developed towards the goal of enabling design of smart networked CPSs with modeling formalisms that lead to analyzable, predictable systems, while promoting learning and analysis on streaming data. To achieve this goal, several aspects of CPS design will be visited throughout the text:

- *Software architectures for data-driven models* - We detail the design process of modular learning and inference techniques within an actor-oriented framework. We study how data-driven workflows can efficiently be integrated with the model-based design process to enable modular design of complex CPS applications.

- *Managing complexity in heterogeneous design* - An essential feature of models, which renders MBD extremely effective for systems engineering is being analyzable and easy to understand. Any platform that enables use of streaming data in model-based design should ensure that these properties are not lost. To maintain modularity and analyzability in models that we introduce, we propose the paradigm of *aspects*. Aspect-oriented modeling is a design pattern for actor-oriented languages, which provides reusable, compositional interfaces to cross-cutting design concerns.

- *Real-time decision making on streaming data* - Domain-specific data-driven control applications, as enabled by the introduced modeling methodologies will be discussed in detail. The primary domains of interest are information-seeking control of mobile robot swarms and smart building applications. Instead of focusing on results on algorithmic accuracy and optimization, we study in detail a general framework that enables domain experts to fine-tune application specific workflows that can benefit from streaming data, while preserving fundamentals of model-based design.

This dissertation consists of seven chapters. Chapter 2 introduces preliminary material on concurrency, distributed computation and an overview of programming paradigms for data-driven design. In Chapter 3, we introduce the paradigm of Aspect-Oriented Modeling (AOM) and present a complete description of the formalism and its applications. This chapter is based on joint work with Patricia Derler, Shuhei Emoto and Edward A. Lee [19]. Chapter 4 presents preliminary material on Dynamic Bayesian Network, limiting the discussion to the statistical analysis of DBNs to the scope of this thesis. Chapter 5 studies the design of data-driven CPS applications using the previously introduced modeling formalisms that are combined with Dynamic Bayesian Networks in the form of an actor-oriented toolkit. Sections 5.2 and 5.4 are based on joint work with Shuhei Emoto and Edward A. Lee [20]. Chapter 6 focuses on the topic of data-driven control of IoT systems, introducing preliminary material on the formalism of control improvisation (CI) and extending the concept with an application scenario in the domain of home automation. The chapter is based on joint work with Daniel Fremont, Rafael Valle, Alexandre Donzé, Edward A Lee, and Sanjit A. Seshia [21]. The final Chapter describes the application areas of the AOM formalism focusing on the multi-robot control domain and concludes the thesis.

The research presented in this thesis has been supported in part by the Multiscale Systems Center (MuSyC) and the TerraSwarm Research Center, centers supported by the STARnet phase of the Focus Center Research Program (FCRP) a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

Chapter 2

Background

2.1 Concurrency and Distributed Computation

The promise of emerging IoT systems heavily depends on the ability to make decisions on streaming data. This requirement poses a fundamental conflict between the traditional data-driven computation models, and the paradigm of streaming data. The fundamental shift in the information source has given rise to numerous developments to support *stream analytics*.

2.1.1 Concurrency Patterns

In traditional software engineering practice, threads have been the dominating concurrency model in computing. However, due to the changing needs of software architecture of emerging complex web and IoT applications, threads have become infeasible for handling many instances of concurrent applications. Threads exhibit highly undesirable consequences including shared mutable state that leads to nondeterminism, deadlocks, and added complexity (e.g., as imposed by resource management requirements of thread pool implementations). Moreover, the nature of shared resources and explicit synchronization management couples the functional code with the implementation, making a threaded program difficult to scale and reason about [75, 94].

As an alternative to multi-threading, event-driven programming paradigms have gained popularity for handling emerging distributed applications. A general concurrency pattern for event-driven architectures is the *Reactor* pattern, which provides a single-threaded model for synchronous and sequential processing of concurrent events [102]. By avoiding many layers of complexity and nondeterminism caused by multi-threading, the Reactor pattern enables event-handling in a more scalable way, which proves beneficial in implementing highly concurrent services such as web servers. A prominent example of a framework that leverages the Reactor pattern is Node.js [10], in which *events* are handled via asynchronous callbacks, while the single-threaded implementation remains idle when there is no event to trigger a callback.

Modern programming languages including Scala, Java, Erlang, and many others offer more than one concurrency model to serve the changing concurrency needs of applications. These patterns include threads, futures/promises, actors, etc., some of which will be discussed in detail in the upcoming sections.

2.1.2 Frameworks for Stream Analytics

Another key programming paradigm that is evolving due to the changing application requirements is the representation of data in computer programs. Data flow from sensors, devices, and web applications is increasingly being modeled as streams, and as a result, more frameworks and libraries are providing capabilities to manage data as such. Many programming languages provide entities to represent (potentially infinite) data streams and operations on streams of data, which include Scala and Java, while specialized programming languages for implementing streaming systems such as StreamIt [109] have also been introduced.

Moreover, with the adoption of cloud based computing and data storage systems, research on higher level frameworks that support computations on data streams has gained momentum. These platforms support operations on streaming data at all levels, from providing distributed messaging systems that also provide some stream processing and storage capabilities (e.g. Apache Kafka [2]) to large scale stream processing frameworks. Notable streaming platforms that support stream processing include open-source platforms such as Apache Storm [5], Apache Spark Streaming [4], Apache Samza [3] and fully-managed solutions including Amazon Kinesis [1], and Microsoft Azure [8].

Apache Samza is a framework that builds upon the core concepts of *streams* and *partitions*. Samza streams are a sequence of immutable messages of a similar type that can have any number of consumers. Streams consist of partitions, which are totally ordered sequences of messages. In terms of data processing, Samza utilizes computational processes called *jobs* that operate on streams and process streams. Jobs are parallelizable via tasks, where each task in a job can process a partition of an incoming stream. Eventually, *dataflow graphs* can be formed using a directed graph of Samza jobs. Figure 2.1 illustrates the stream processing workflow in Samza.

The core abstraction in Apache Storm, similarly, are data streams. Storm introduces the concept of Tuples, that are individual events that sequentially add up to potentially unbounded streams. One differentiating factor of the Storm architecture is the concept of a *spout*, which is essentially a source for a data stream. This abstraction makes it easy to integrate multiple sources of data into the framework, and additionally, the notion of *reliable and unreliable spouts* makes it simple to declare

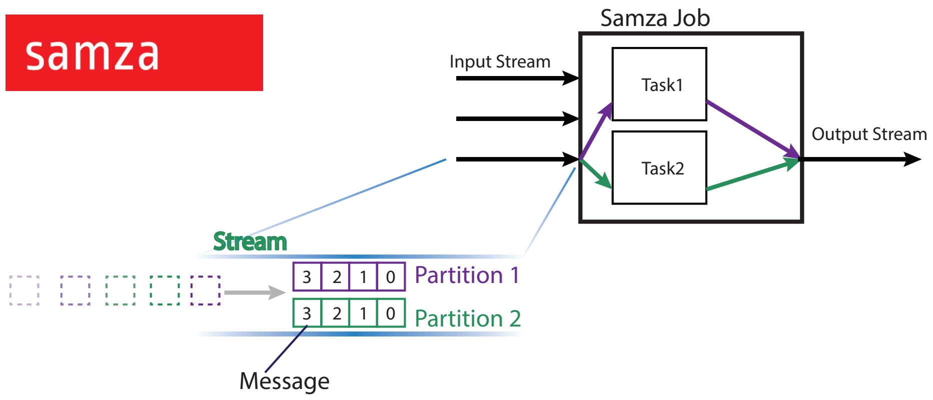


Figure 2.1: Stream Processing in Apache Samza

whether spouts provide resilience in the event of a tuple failing to be processed. Similarly to Samza, Storm introduces processing units called Bolts, that are responsible for performing stream transformations. Storm also supports dataflow graphs that incorporate sources (spouts) and bolts, connected in a directed graph [104].

The main contrast of Spark Streaming from the previously explained platforms is that the stream processing is not done on a per-event basis in Spark Streaming, but on micro batches of data. A live stream is divided into micro-batches called *discretized streams (DStreams)*, preprocessed by the Spark Streaming engine and sent to the Spark engine for further computations.

Investigating the stream processing of notable frameworks reveals that modern streaming computation builds upon a number of common concepts. The abstraction of streams and processes that operate on streams will be of key interest for the formalisms to be introduced as part of this thesis.

2.2 Programming Paradigms for Data-Driven Design

2.2.1 Object-Oriented Programming

Object-oriented programming (OOP) has been one of the most widely-used programming paradigms since its introduction with the language Simula 67 [35], and has been supported by many programming languages ever since. In OOP, programs are built out of *objects with well-defined interfaces* that interact via procedure calls. This paradigm replaced the early programming languages, which relied upon many small modules that are controlled via global variables, that are accessible by the entire program, with vague scoping rules. Using objects created an extremely desirable layer of abstraction that separated external code that consists of function calls to object methods from internal code (object's internal definition), and eliminated the need for global variables in most cases [56]. In OOP, data and functions to access data are encapsulated within the object, and the internal implementation is hidden. Code duplication is reduced by inheritance, which enables common code reuse in related objects. OOP enables breaking down a complex problem into simpler sub-problems in terms of abstract data types and their interactions .

Despite being a great advancement towards scalable and less error-prone software development, OOP still suffers from fundamental limitations. We will discuss some of these problems, specifically in the context of CPS development.

Although OOP provides abstraction and modularity in design, it does not contribute to representing concurrency. Using a call-return semantics does not provide a structured means of modeling concurrent computation, and modeling concurrency in OOP often resorts to multi-threaded code with ambiguous semantics.

Another essential programming challenge arises when building complex object-oriented systems with multiple *cross-cutting concerns*, which often need to be implemented in a modular and reusable way as to provide understandability and ease of maintenance. The issue of modeling cross-cutting concerns is amplified by the object abstraction, since objects are optimized to be self-contained units isolated from their environments, revealing only a subset of their functionality via available public procedures. This model falls short of expressing design aspects that possibly extend beyond a single object's functionality, such as logging, policy enforcement, and optimization.

2.2.2 Actor-oriented Modeling

A modularization technique that complements objects is actor-oriented design, which is a high-level design paradigm that aims at providing concurrency and scalability [60, 17, 81]. Actors are components that execute concurrently and communicate via messages sent and received through ports, in contrast to the procedure calls commonly used in OOP. The actor model provides a clear abstraction against shared mutable state, since all state mutation is done via messages passed between actors. By doing so, actors provide better scalability, avoiding threads and locks, and exhibits many other desirable properties to be discussed throughout the text.

Actors are components -that, in fact, can also be objects- which execute concurrently and communicate via messages. Many of the model-based design tools including Ptolemy II [96], Simulink [14] and NI LabVIEW [9], as well as application frameworks such as the Vert.x framework [15] and the Akka library [122] support actor-oriented development.

The semantics of actors and the communication between them can vary widely across domains. A concurrency model and communication strategy for actors form a concurrent *Model of Computation* (MoC), and distinct MoCs can be combined to create heterogeneous models [96].

Based on the actor model, components called *accessors* have been introduced.

The accessor model is based an abstraction where sensors, actuators and (typically remote) services are be wrapped by an actor interface [74]. Accessors leverage an event-driven architecture, where events are internally handled via atomic asynchronous callbacks.

Chapter 3

Aspect-Oriented Modeling

In this chapter, we introduce the aspect-oriented modeling paradigm that enriches the actor model by introducing mechanisms to handle cross-cutting concerns in CPS design.

One of the biggest challenges in cyber-physical system (CPS) design is their intrinsic complexity, heterogeneity, and multidisciplinary nature. Emerging distributed CPS integrate a wide range of heterogeneous aspects such as physical dynamics, control, machine learning, and error handling. Furthermore, system components are often distributed over multiple physical locations, hardware platforms and communication networks. As the level of complexity and heterogeneous concerns to be considered in design time increase, no single design pattern succeeds in simultaneously handling the many aspects of complex systems. While model-based design (MBD) has tremendously improved the design process, CPS design remains a difficult task. Models are meant to improve understanding of a system, yet this quality is often lost when models become too complicated.

In the previous chapter, we introduced OOP, discussed some important design concerns that arise when modeling complex CPS using OOP, and proposed actor-oriented design as a solution to modeling concurrency between components. Despite the tremendous improvements introduced by the actor model, the issue of addressing cross-cutting concerns into the design process has not been resolved.

We show how to use aspect-oriented (AO) modeling techniques in MBD as a systematic way to segregate domains of expertise and cross-cutting concerns within the model. As a motivating example throughout the thesis, we will discuss several design concerns that arise when building multi-robot sensing and control applications. We will illustrate the use of AO modeling techniques to manage complexity, while introducing modular design components to handle streaming data for adaptive development and design-space exploration for these complex CPSs.

3.1 Design Paradigms for CPSs

Cyber-physical system design integrates a wide variety of heterogeneous disciplines, including control engineering, mechanics, thermodynamics, sensors, electronics, networking, and software engineering [76]. Engineers use domain-specific tools and techniques in each of these disciplines, but integration of the diverse tools and techniques remains challenging [48]. MBD [67] has proven successful in several of these domains, including, for example, modeling and simulation of physical dynamics using Modelica [111], design, simulation, and code generation of control systems using Simulink® (by MathWorks), and design of instrumentation systems using LabVIEW® (from National Instruments). In network simulation, MBD is adopted in modeling and simulation of communication networks using OPNET Modeler® (by Riverbed) and ns-3 (`http://www.nsnam.org/`). Moreover, architectural system modeling benefits from MBD, as exemplified by the unified modeling language (UML), the architecture analysis and design language (AADL), and OMG SysML [62].

Despite the domain-specific benefits of MBD, integrating these tools and techniques, remains a daunting challenge [18, 66]. Accidental complexities such as incompatible data representation, lack of open-source APIs, and vague or unspecified semantics dominate, so that it uncommon to achieve effective integration.System integration ends up happening late in the design process, when prototypes of all components are available, and problems that emerge at that stage can be very expensive to fix. As industrial CPS systems get more complex, it becomes necessary to provide greater assurance of safety, scalability, and reliability from the early design stages. This demands interoperable, verifiable, modular, and open interfaces that simplify the task of integration. In this chapter, we describe an approach that focuses on integration in the layer of modeling methodologies. As opposed to the usual meaning of *integration* in the level of tool integration, we will focus on adapting actor-oriented abstractions of design aspects, which can later be composed as domain-specific tools.

In section 2.2.1, we discussed that OOP often suffers from expressing crosscutting concerns in design. This problem fundamentally arises from the object encapsulation, which isolates objects from the context it is created in, and intentionally limits the environment's access to the object's internal mechanism other than what the object reveals by its access methods. Despite providing better data encapsulation and an explicit semantics for concurrency, actor-oriented design still suffers from the same issue. Many requirements of reliable CPS, including safety, security, verifiability, adaptability, and platform independence are usually very difficult to address using a monolithic actor model.

Even in the case when actor-oriented design is used to address many design concerns, the models often end up tangling many cross-cutting functionality, unintentionally modeled in a way that can not be easily factored out. Although CPS are intrinsically complex, the models need not be equally complex to successfully capture essential aspects of the systems. The goal of the following formalism is to improve the degree to which a model can accurately reflect the complexity in system design, without making the models themselves so complex as to degrade their utility.

3.2 Aspect-Oriented Programming

In the late 1990s, Kiczales et al. introduced aspect-oriented programming [70] to resolve OOP's inability to express the aforementioned issues relating to cross-cutting functionality. Some common *concerns*, which are defined to be a functionality or specification to be ensured throughout the system design, can be implemented in a modular way using the aspect paradigm.

Aspects have been embedded in programming frameworks such as AspectJ [69] and Spring [64], which extend Java and other OOPs with specific syntax supporting aspects.

AO programming has some clear benefits. Code tangling, code duplication, and scattered code can be reduced through abstraction and modularization. However, the adoption of AO programming has been slow. Studies on fault-proneness of aspect-oriented programs [46] show that the *obliviousness* property in AO programming causes faults due to lack of awareness among base code and aspects. Moreover, debugging aspects can be difficult [123].

An *aspect-oriented program* identifies cross-cutting concerns in the software design and defines modular aspects to implement these in a way that is separated from the core functionality. Figure 3.1 illustrates the conceptual differences between an object and an aspect-oriented implementation of an aspect. Consider, for example, a *logging* aspect. Monitoring and recording certain execution details of a program is clearly not part of the core functionality, and in fact, a typical logger would be subject to change as the core functionality evolves.

An aspect-oriented program consists of several entities. We will briefly explain the terminology relating to these entities and will make analogies with aspects in actor-oriented models in the following sections.

- *Weaving* is the process of linking aspects with the application, which can either happen at compile time or at run time.

- *Crosscutting* is the implementation of the weaving rules. This is performed by the compiler in object-oriented aspect implementations. The crosscutting can either be static or dynamic. A static crosscutting is an aspect implementation that relates to a class, interface, or another aspect, and does not directly modify the execution. A dynamic crosscutting is essentially the weaving of new behavior into the program with the help of an aspect. A crosscutting consists of multiple elements

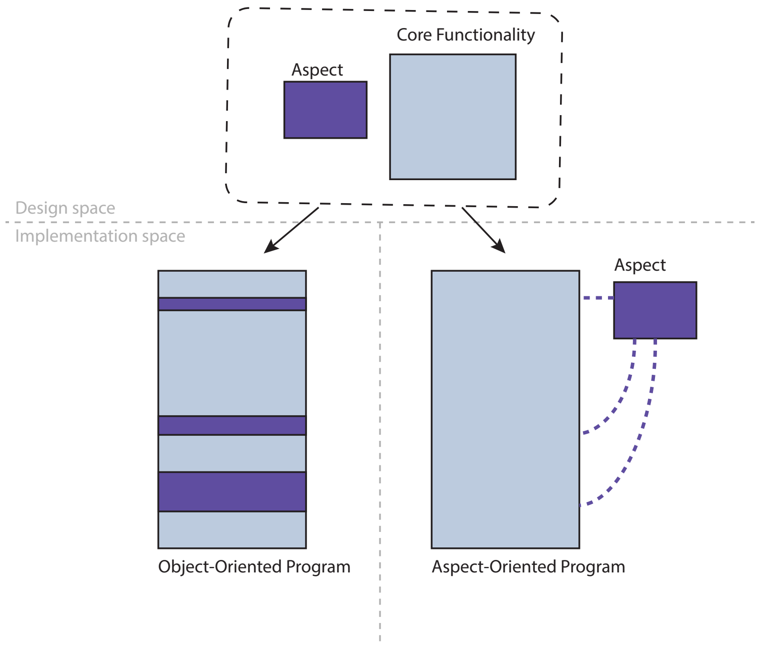


Figure 3.1: Implementation of a cross-cutting concern in OOP and AOP

  - *Join point* is a point during the execution where the program interacts with an aspect. In AO programming languages, this often refers to a method execution, which weaves the crosscutting actions into core functionality.
  - *Pointcut* is a program construct that identifies join points and builds context around these. Pointcuts can be thought of as specifiers of weaving rules, where join points are instances that satisfy these rules.
  - *Advice* is an action taken by an aspect at a particular join point. An advice can be configured before, after, or around the join point.
  - *Introduction* is a static crosscutting, that introduces changes to the entities being affected. An example introduction can be a method that is added to a class by the aspect.

- *Aspect* is the basic building block of aspect-oriented code, which contains the code that specifies rules to weave the cross-cutting concern with the code functionality [73].

3.3 Types of Aspects in CPS Design

The separation of the core functionality from the cross-cutting ones by design paradigms is not specific to OOP, but a more general paradigm that in fact becomes much more essential for developing cyber-physical systems. The following sections will discuss how the ideas introduced by AO programming can be applied not only to OO design, but also to actor-oriented models. We focus on models in the CPS domain that are highly heterogeneous in nature, causing them to become hard to validate and verify due to the inherent complexity and added peripheral requirements.

It is common to use the term "functional model" for a model that describes the core intended behavior of a system. For example, a functional model of a cooperative cruise control system may describe the feedback control laws and the physical dynamics of a car. So called "non-functional" aspects might include properties of an implementation such as communication latencies, faults, and energy consumption. Of course, what is viewed as "core intended behavior" will depend on who is building the model. But generally, when an engineer builds a model, that engineer has some view of a "core intended behavior" and some notion of other concerns that may be important, but are distinct from the core functionality. Our goal is to treat these "non-functional" concerns as aspects, specifically to be able to model these without entangling them with the core intended behavior. Examples of such concerns that arise in CPS are as follows.

Communication. CPS applications are often distributed, sometimes widely. Communication infrastructure is important, and communication artifacts such as delay and losses must be taken into account during system validation. Resource contention and communication delays can lead to sub-optimal system behavior that is hard to predict and is not reflected in a purely functional model. Modeling of networks, however, is itself a sophisticated domain, therefore, entangling network models with core models is undesirable. Aspect-oriented modeling of networks in cyber-physical energy systems is discussed in [23].

Execution time. A model of "core intended behavior" will typically not include execution time of software components. At early stages of a design, this cannot be known, as it requires considerable implementation detail. Even at later design stages, complexity of the underlying architecture can make it difficult to precisely account for execution time [117]. Modeling execution time is again a sophisticated domain, and such models should not be entangled with core intended behavior.

Architecture. Design choices such as hardware architecture (multicore computers, centralized triple-redundant machines, distributed microcontrollers, etc.), scheduling strategies (time triggered, earliest deadline first (EDF), etc.), and mapping of software functions onto hardware resources can profoundly affect system cost and behavior. If we can avoid entangling architecture models with core intended behavior, then we can facilitate design space exploration at different levels of abstraction [101].

Error handling. Error handling is necessary in real systems, but the additional logic can clutter the core model. Aspect-oriented modeling helps in separating error handling logic from the core model. In addition, proper error handling requires fault models, and testing the system under fault conditions can also be handled through aspects.

Logging and debugging. Logging and debugging features are often heavily used during development and less in a deployed system. Factoring out the logging and debugging infrastructure makes it easier to include or exclude.

Verification Checking correctness of the intended system behavior is an essential aspect of safety-critical CPS. Aspects can be used to automatically check compliance with formal requirements during system development, and to detect fault conditions that result in specification violations in a deployed system. Recent work on design contracts [93] formalizes high-level system requirements, that can be implemented as design *aspects*.

Fault modeling and anomaly detection System modeling primarily aims at representing the nominal behavior of a system. In many CPS domains such as automotive, aerospace, and manufacturing, fault models are an essential part of system development. Faults and anomalies that could occur in a system can be factored as aspects. This naturally segregates faulty behavior from the nominal system workflow, and enables efficient modeling of cascading faults.

Security concerns Wasicek et. al. [115] present aspect-oriented modeling as a model-based design technique to assess the security of CPS. By associating attack models with the CPS in an aspect-oriented manner, the designer can gain insights into the behavior of the CPS under attack.

Dynamics and sensing models The foundation of many CPS aspects such as control, machine learning, and optimization build upon mathematical models of a physical process. Consistency of the common mathematical models across the system becomes prone to errors due to the composition of components from different areas of expertise. Using aspects enables these shared mathematical models to be explicitly factored out, and reused by multiple parts of the CPS model.

Environmental Constraints While the *functional* system model aims at representing the core intended behavior of a system under a set of environmental conditions, additional constraints can be enforced on system operation in run time. For example, considering a CPS that consists of mobile robots, the physical space (often represented by a 'map' model) in which the system operates is a type of environmental constraint that can be thought of as an aspect. A map becomes a cross-cutting concern for many parts of the functional model, potentially affecting communication, dynamics, control, as well as noise models.

In the next section, we will describe in detail an infrastructure for aspect-oriented modeling (AOM) that we have realized in the Ptolemy II framework [96]. The techniques presented in this chapter should be easily applicable to many commonly used actor-oriented design tools.

3.4 Motivating Example

Design of industrial cyber-physical systems is a complex task due to the intrinsic complexity and timing constraints of these systems. To date, distributed system design has been prone to errors due to concurrency, timing and heterogeneity of system components.

Emerging industrial CPS are increasingly using autonomous robotic systems. Design of cooperative behavior given application constraints such as dynamic capabilities of robots, resource conflicts, and environmental constraints is an active field of research.

Industrial applications, including factory automation, assembly, monitoring, and disaster response rely on cooperative behavior of humans and machines. Robotic swarms, which extend the concept of cooperative machine intelligence, fit well into this framework. "Swarm intelligence" is defined as a system property that arises from the interaction of non-intelligent mechanical robots to collectively form an "intelligent" system [31].

One example in an industrial setting is a disaster response scenario, where the consequences of a disaster may have rendered a physical space hazardous for humans to explore. In addition, parts of the site may not have ground access at all, due to debris caused by the disaster. Collaborative mapping of earthquake-damaged buildings is an example of this, where ground and aerial robots need to work together to create a map of the hazardous site [90].

In addition to exploration and risk assessment, robots can also perform mechanical tasks in sites that are inaccessible to humans. The Fukushima Daiichi nuclear reactor cleanup and investigation effort Japan provides such a scenario. Robots are currently being deployed inside the reactors to collect critical information on radiation levels, as well as on safe paths to be taken in subsequent missions [113]. Larger robots are being employed to carry out mechanical tasks within the reactor, such as to deploy vacuum and filtration systems that reduce nuclear contamination and allow human operators to safely enter the site [112].

Motivated by such scenarios, we illustrate the concept of aspect-oriented modeling on the design of such robotic swarms. A simplified top-level model is shown in Figure 3.2, where two sets of robots are deployed: (i) a team of lightweight *observation robots* that are equipped with sensors whose task is to cooperate to assess an environment in order to locate a target in the presence of potential hazards, and (ii) a heavy-duty *main robot*, with limited maneuvering abilities, whose task is to retrieve a target from an unfamiliar environment.

In the model, a hierarchical control strategy is being deployed to fully enable swarm intelligence. First, an observation-optimizing controller receives sensor measurements from observation robots, estimates the position of the target based on the robot dynamics and sensor readings, and steers the observation robots, subject to dynamics and safety constraints, to reduce *uncertainty* in target position in the subsequent control steps. Meanwhile, a higher level controller receives the estimated target position, and steers the main robot towards the target. The sensors, of course, have limited sensing capability. For example, in our model, we assume that they can only make imperfect measurements of their distance from the target.

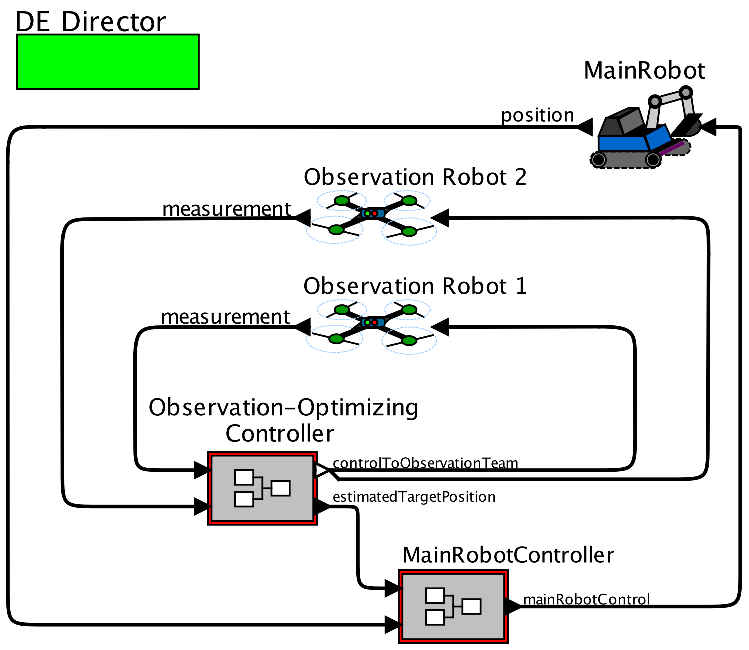


Figure 3.2: A model of a robotic swarm, cooperating to carry out a target retrieval task within an unfamiliar environment

The modeling environment used here is Ptolemy II [96], which is a framework for building and simulating actor-oriented models of heterogeneous systems. A Ptolemy model consists of actors that communicate via ports. The semantics of the communication, also referred to as the model of computation, is given by so-called *directors*. Different MoCs can be combined hierarchically to represent heterogeneous systems.

The robots and controllers are composite components, where a single icon represents a potentially complex submodel. The components communicate via time-stamped events, using the discrete-event MoC, represented in the model by the `DE Director`.

Such a model, which captures component interactions alone, is often created by a system designer for evaluation of functional behavior and timing analysis of the application. Once the behavior has been evaluated and deemed suitable, the next step is deployment, during which the timing behavior of the application might change. For example, in deployment, the communication between controller and the robots will be subject to context-dependent latency, packet losses, and re-ordering of messages.

One possible model that includes the communication aspects is given in Figure 3.3. In this model, the `Network` actor represents the shared communication resource. It is a hierarchical component that can include delays, losses, and re-ordering. Such a hierarchical component could include sophisticated models of specific networking

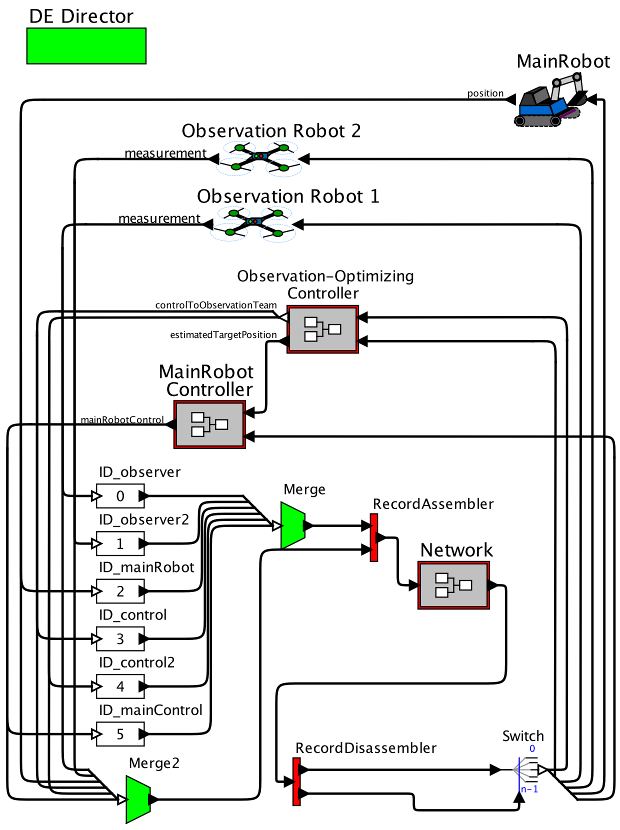


Figure 3.3: A poor model of a robotic swarm with network specifications.

technologies, including, for example, WiFi interference and MAC (media access) protocols.

The model in Figure 3.3, however, is awkward. In order for the communication aspects to affect all relevant communication paths, the model builder is forced to model the construction of packets that include addressing information and to multiplex these packets through a single model of the communication fabric. The designer is often not interested in such low level modeling details but only in the high-level effect of the network on application behavior. Moreover, the logical communication paths of the original models have been lost, eliminating many of the advantages of a visual modeling syntax.

Nevertheless, a Monte Carlo simulation, yielding results such as that in Figure 3.4, can be used to study the behavior of the system with and without network models. These results illustrate the detrimental effects of packet losses and latency, and an engineer can determine thresholds on networking behavior that would put the target retrieval mission at risk.

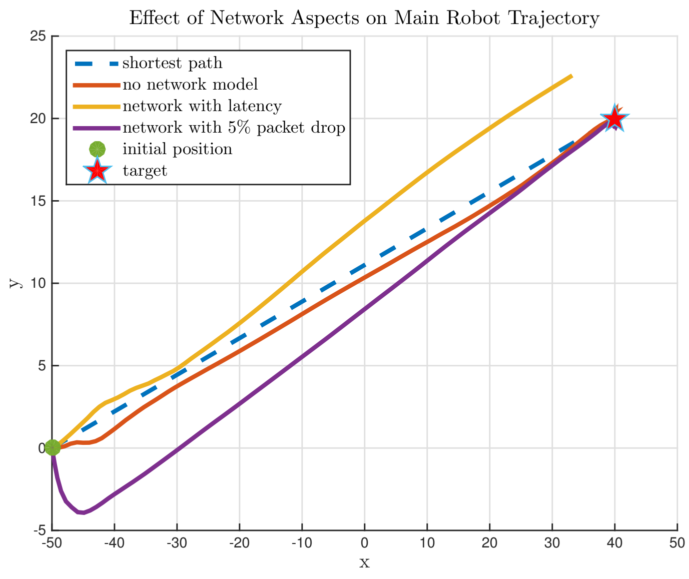


Figure 3.4: Simulated effect of network aspects on main robot trajectory.

It is important to observe that the network fabric used to enable communication between components is not an intrinsic part of the system design; it is only an implementation choice. Therefore, the communication via a shared resource can be considered an *aspect* of the system and can be modeled as such. An aspect-oriented alternative that uses our Ptolemy II extensions is shown in Figure 3.5. The network aspect, represented by an icon at the top, models the delay and resource contention of communication; connections that use the network, namely, the channels between robots and controllers, are annotated with textual parameters that bind those communication links to the network model. This preserves the benefits of the visual syntax, in that, the new model still visually represents logical communication paths. But more importantly, it abstracts away the low-level details of the network implementation, such as packet structure and addressing. These were needed in Figure 3.3 as an accident of the modeling technique, whereas in the aspect-oriented alternative, the low-level routing functionality is handled internally by the aspect. The aspect illustrated here is a *communication aspect*. There are many additional aspects that can be modularly incorporated into a model in a similar way, as will be explained next.

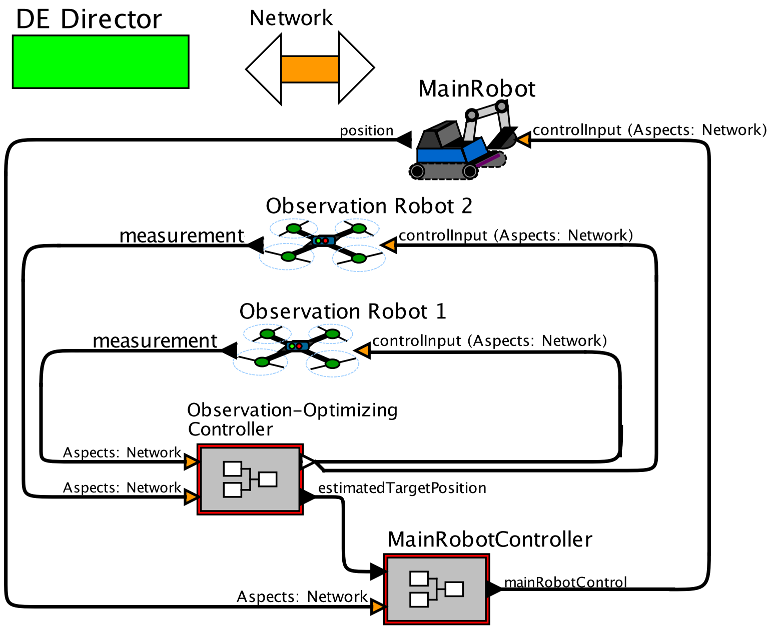


Figure 3.5: A model of a robotic swarm with network specifications modeled as aspects

3.5 Aspect-Oriented Modeling (AOM)

The basic ideas of actor-oriented programming have been introduced in section 2.2.2, and presented in [19]. In analogy to aspect-oriented programming, which separates *aspects* to be encapsulated portions of code that interact with the core functionality using procedure calls, an actor model can also be annotated with additional information that is orthogonal to the information in the model. This information can be used to statically evaluate the model or to modify execution. An example would be evaluation of the cost of implementing a system. By annotating each model

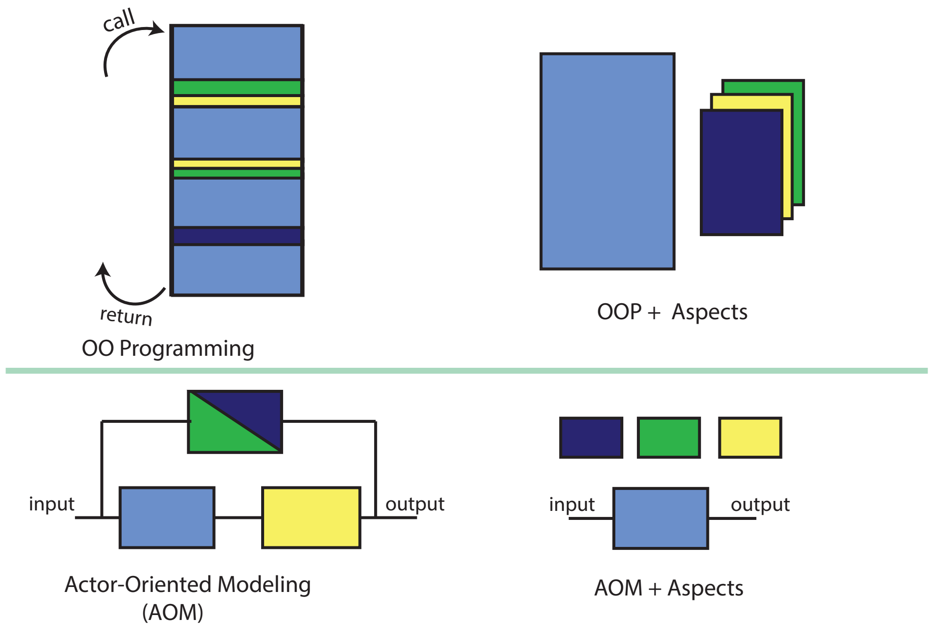


Figure 3.6: Comparison of Design Patterns

element with a *price*, the cost of the entire system can be computed in design time, and updated dynamically when subcomponent prices change. The cost can also be continuously monitored and compared to a constraint, for instance, an upper bound on cost, raising a flag when the upper bound has been exceeded.

With aspect-oriented modeling, we will go a step further than open-loop monitoring. We will annotate models with information that is evaluated dynamically and can in turn change the system behavior. An example is the communication aspect described in the previous section.

AOM aims at providing a mechanism to better represent orthogonal concerns in actor-oriented models the same way aspect-oriented programming does in object-oriented programs. The four modeling paradigms are conceptualized in Figure 3.6. We will now formally tie concepts from aspect-oriented programming and explain how they differ from their object-oriented counterparts.

An *advice*, in the scope of this work, represents an actor that implements the cross-cutting concern. In the previous example, the advice is the `Network` actor.

A *join point* is the execution of an actor or the transmission of a token. While advice actors do not encode any information about the model they are used in, they encode information regarding at which join points they can be used. For instance, the `Network` aspect can only be associated with connections between actors. We generalize this to input ports on actors, meaning that if the `Network` is associated with an input port, the incoming connection is enhanced by the *advice*. The fact that *advice* actors can only be used on certain join points can be utilized to guide the model builder. In Ptolemy II, upon instantiating an advice actor in a model, parameters are added to all the possible join points in the model, at which the use of the advice can either be enabled or disabled by the model builder.

A *pointcut* is a join point where the use of the advice is enabled. An *aspect* is then all enabled join points together with the advice.

Ptolemy II handles the *weaving* at run time. A simulation framework where code is generated before simulation would need to include the weaving in the generated code.

In many aspect-oriented programming languages, pointcuts rely on naming conventions to find the points in the program where aspects should be executed. Thus, changes to the program may easily break the aspect integration, a consequence often referred to as *the fragile pointcut problem* [106]. In the newly introduced paradigm of aspect-oriented modeling, advices need not be aware of the model and pointcuts that are defined. Due to the actor-oriented nature of programs, deleting an advice from the model automatically results in a deletion of all pointcuts, overcoming the unintended consequences originally introduced by aspect-oriented programming.

In this chapter, we investigate aspects with two types of pointcuts: on input ports of actors and on actor executions. An aspect with a pointcut on an input port of an actor executes the advice whenever a token is sent to this input port. The advice can modify the token (e.g., probabilistically dropping it) or perform other operations. In the previous example, the `Network` advice implements resource contention and packet drops on tokens.

Figure 3.7 illustrates the concepts introduced on an abstract example. The model contains 5 actors, A_1, A_2, A_3, c and e, where c and e are advice actors. Actors A_1 and A_2 communicate as illustrated by their connections. The advice c can be enabled on communications between actors, thus the communication between A_1 and A_2 as well as the communication between A_1 and A_3 form join points. In the example, the advice is only enabled on the communication between A_1 and A_2, which describes a pointcut. Advice actor e can be enabled on actor executions, so its join points are actors. In this example, e is enabled on actor A_1 and actor A_3, but not on A_2. In the figure, enabled join points are illustrated by highlighting.

3.6 Aspect-oriented modeling in Ptolemy II

To implement join points and pointcuts in Ptolemy II, we use the decorator pattern [54], which allows adding behavior to an individual object without affecting the behavior of other objects of the same class. We implement advices as *decorators* that decorate join points with additional attributes.

Figure 3.8 illustrates the implementation of advices that decorate communication and execution between actors in Ptolemy. A communication advice decorates all input ports in the model with a boolean attribute *enable*. If this enable flag is true, the

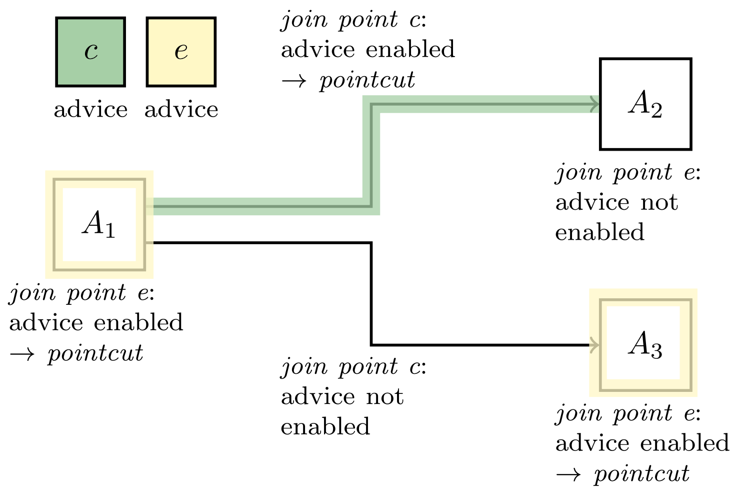

c
advice
e
advice
join point c: advice enabled → *pointcut*
A_2
join point e: advice not enabled
A_1
join point e: advice enabled → *pointcut*
A_3
join point c: advice not enabled
join point e: advice enabled → *pointcut*

Figure 3.7: Aspects in actor-oriented models.

receiver of the port is wrapped by an intermediate receiver, which intervenes in the communication and coordinates with the aspect actor. Communication advices can be composed serially. In this case, the original receiver is wrapped by multiple cascaded intermediate receivers, providing associations with multiple advices. The order in which communication aspects are enabled can be controlled by the model builder. In the figure, the weaving of a communication actor is performed in the following steps:

1. A source actor (`A1`) intends to send a token to the destination actor (`A2`), where the token is originally destined to the input port of the destination actor.

2. Token is initially delivered to the intermediate receiver.

3. The intermediate receiver delegates the token to the input port of the actor that implements the *communication advice*.

4. Upon being processed by the aspect implementation logic contained in the advice, the token is routed back to the original port it was destined to.

5. The input port delivers the token to the destination actor to be further processed.

The implementation of an *execution* advice (vs. a communication advice) is also illustrated in Figure 3.8. In contrast to the communication aspect, an execution aspect decorates *actors* (vs. ports) in a model with an *enable* flag. If this flag is set to true, the advice actor will be consulted each time the actor is executed. In Figure 3.8, the Actor that has been associated with the `ExecutionAspect` by setting this flag is highlighted in blue. This association is implemented by modifying the

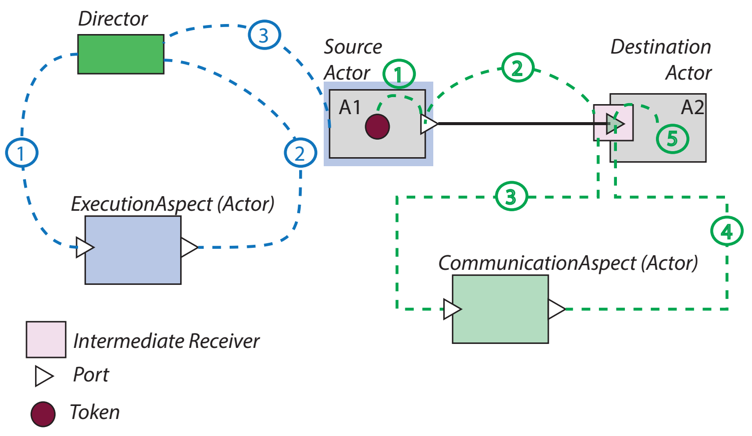


Figure 3.8: Execution and Communication Aspect Semantics

director. The director, in absence of an execution aspect, (a) selects the next actor to be executed and (b) executes the actor by calling its fire function. Association of an execution aspect adds an intermediate step to this two-step operation between steps (a) and (b). Here, a call to the advice is inserted upon selection of an actor that is decorated by an execution advice. In case the advice decides that the actor cannot be executed, the director can choose another actor to be fired. Note that if the Director implements a timed MoC, the authorization for actor execution may become a function of the time step. Consider, for example, an execution aspect that implements an architectural concern. If the architectural resource that is emulated to be processing the actor execution is busy at the requested time step, the execution aspect will continue to reject this actor's execution until resources become free at a future time.

Weaving, the process of linking aspects with the application, is performed by the Ptolemy II runtime. Part of the weaving is implemented by the director that executes the model. Note that neither communication nor execution aspects have physical connections in the graphical actor model. This feature, by design, enables aspects to remain *orthogonal* to actor execution visually, as they intend to represent logically orthogonal concepts.

3.6.1 Atomic and Composite Aspects

An aspect is an actor that implements the decorator interface. Atomic aspect actors contain the entire implementation in a single body of source code. Composite aspect actors are models themselves. A composite aspect contains a director and special actors that receive requests and produce responses. Composite aspects enable much more sophisticated models, for example of communication networks. A model element that is decorated by a composite aspect receives an additional parameter, the request port. This is the name of the actor contained in the composite aspect that receives the request. When the request port actor receives the request, it produces a token that contains the request actor or receiver together with other attributes that are specified on the port. For instance, additional attributes could be execution times of actors.

3.7 Related Frameworks

Since AO programming is often considered an extension to OOP, various AOM extensions to object-oriented languages have been proposed. Naturally, UML-based AOM design approaches have been developed [120]. While some approaches provide general-purpose AOM languages, others only focus on specific aspects. Our approach is a general purpose AOM language in that it allows the definition of arbitrary aspects. Some work on UML AOM extensions is related in that they focus on CPS aspects that we also explore here. For instance, Liu and Zhang [86] present an aspect-oriented framework that combines UML profiles and real-time logic (RTL) for specifying QoS properties such as timing, reliability, and safety. Recent work on using AO modeling for automation systems is presented by Wehrmeister et al. in [116]. Espinoza et al. [44] describe annotations of schedulability and performance analysis data in UML models. Mechanisms similar to the ones used to weave aspects in model-based design environments are introduced in Gray et al. [57], where AOM is used for domain specific, cross-cutting constraints.

The main differentiating factor between the presented AOM formalism and UML based AOM extensions is the purpose of the models. This chapter focuses on the design of executable models, i.e. models have a clear, deterministic, and concurrent semantics, whereas UML models usually do not have execution semantics.

In relation to system design, the closest concept that have been introduced in literature is part of the framework Metropolis [27, 125]. The Metropolis platform focuses on separating the behavior and the performance aspects of architectural models. Here, so-called *quantity managers (QMs)* are used to assign *quantities* to events, which in turn are scheduled by the framework. The concept is similar to the introduced AOM mechanism in that common aspects are abstracted away from the functional model and added later via quantity managers and schedulers. The Metropolis project facilitates platform-based design [101], which enables co-design of functional and architecture models that are evaluated as a whole. QMs in this context have been applied to modeling execution costs, timing, energy constraints, and scheduling policies. Data-driven modeling using QMs have not been explored to date.

AADL, the architecture analysis and design language (formerly known as avionics architecture description language) [45] aims at modeling and analyzing complex architecture models. Architectural descriptions can be extended with annexes, such as the behavior annex that allows a state machine description of component implementations, or the error annex, which supports fault modeling. De Niz et al. discuss different aspects and the separation of concerns, in particular nonfunctional concerns, in AADL [36]. AO4AADL [88] is an aspect-oriented extension to AADL to master complexity and ensure scalability.

3.8 Conclusion

This chapter introduces a modeling paradigm called aspect-oriented modeling (AOM) and discusses how AOM facilitates managing the complexity of actor-oriented CPS models by enabling development of composable models for cross-cutting concerns. In this chapter, the execution semantics of aspects with an emphasis on their prototyped implementation in the Ptolemy II framework has been highlighted. Chapter 7 will further explore the application areas of AOMs that include fault modeling, monitoring, as well as contract and execution models. Additionally, a more comprehensive case study which illustrates the use of aspects in an industry scale smart grid application has been presented in [23].

Chapter 4

Dynamic Bayesian Networks

4.1 Statistical Models for Data Sequences

The previous chapter discussed design patterns that enable structured development of data-driven systems given heterogeneous cross-cutting concerns. Having proposed a software architecture for modeling such concerns, we now focus on the algorithmic side of implementing aspects for data-driven system design. The focus of this chapter will be to introduce a family of algorithms that synthesize observed data and statistical models of underlying processes to enable dynamic decision making.

Sequential data has been of interest to many fields of science and engineering. A very common scenario for data sequences is to consider data that has been obtained at different points in time from the same source. Such sequences are often referred to as *time-series* data. A very rich literature on statistical modeling and inference on such sequences exist, which focuses on answering questions on how samples within the sequence relate to each other.

The classical approach to time-series analysis is centered around capturing the correlation between samples of data that have been collected at adjacent steps in time, with the assumption that a particular time-series sequence is a sample realization of a stochastic process. For this purpose techniques that model the current value of a time-series as a parametric function of past and present samples have been developed. One specific approach in this family of techniques are autoregressive integrated moving average (ARIMA) models [29], that prove very effective in capturing seasonality and trends in time-series data, and have been used extensively in forecasting applications [105].

When working with datasets alone, that is, batches of data that present some observed quantities as a function of a dependent variable (usually a physical quantity such as time, location, or an enumeration over a finite set) these models prove extremely useful. Despite being very effective in modeling data as a function of its own (past and present) samples, time-series models lack the crucial ability of tying a data sequence to a physical process that generated the data in the first place. For instance, a historical dataset such as yearly average global temperature deviations collected over several decades can effectively be modeled as a time-series, and an ARIMA model fit can be obtained to explain trend, seasonality and a residual noise component in the data. In such a model, it is irrelevant to consider a physical model of earth's surface and investigate how the data relates to the physical system, the focus is on capturing the variations in the *data* itself, and how they relate to the previously collected samples.

Many applications that depend on data collected from a sensor in a CPS naturally extend far beyond such time-series analysis. It is often of interest to incorporate prior information about physical processes, keep track of several quantities that might be affecting each other, as well as the statistical characteristics of observed data. Specifically, when reasoning about what a data stream implies about a system, the formalism should focus on the *underlying cause* of the observed data, with the eventual goal of inferring a change in the indirectly observed process. Prediction follows a similar pattern; instead of trying to predict future samples of data given a window of previous ones, the focus is on predicting future states of the system given data, by first inferring the most likely underlying states of the system that may have produced the observed data samples, and using this inference to predict future states.

A more contemporary way of modeling data streams is to use a family of models known as Bayesian Networks. Bayesian networks are a class of probabilistic graphical models that allow modeling random variables and their conditional dependencies, enabling inference and learning tasks to be parameterized over random variables as opposed to just over data samples. The subclass of Bayesian Networks in which sequences of random variables are expressed (often as a function of time) is known as Dynamic Bayesian Networks (DBNs). Due to the ability to associate an observed data sequence to a graph of *hidden random variables*, DBNs are not subject to limits on expressing relationships between variables, which classical time-series models suffer from. Additionally, DBNs are not subject to finite-window effects, and can incorporate prior information, as well as multivariate relationships between inputs and outputs [91].

One of the most widely used tools for inference and prediction on streaming data streams are Kalman Filter models, used in many control and estimation applications in the domains of linear system theory and signal processing. Another popular model that can efficiently model changing statistical nature of data sequences are Hidden Markov Models (HMMs), which assume a hidden state space that evolves according to a Markov model. In an HMM, The Markovian dynamics of state transition is only observable through measurements that are a function of the *hidden* state. Both Kalman Filter models and HMM variants are special cases of DBNs. An incomplete list of subclasses of DBNs that are of interest to this thesis have been illustrated in

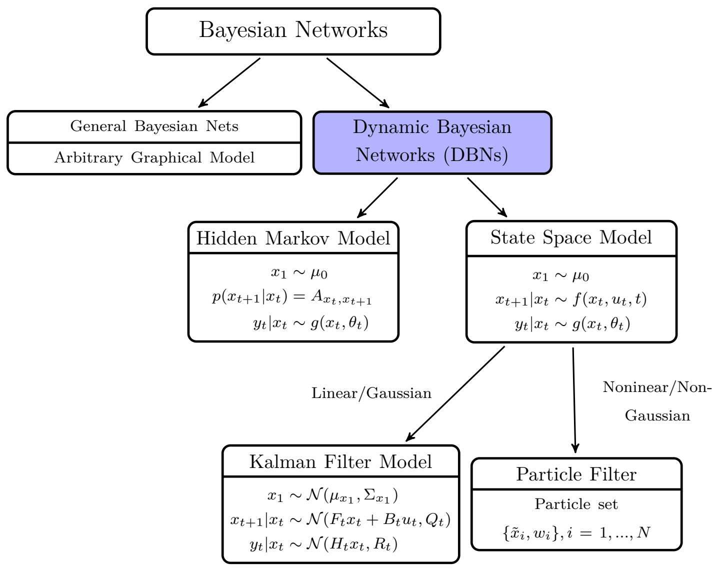

Bayesian Networks
General Bayesian Nets
Arbitrary Graphical Model
Dynamic Bayesian Networks (DBNs)
Hidden Markov Model
$x_1 \sim \mu_0$
$p(x_{t+1}|x_t) = A_{x_t,x_{t+1}}$
$y_t|x_t \sim g(x_t, \theta_t)$
State Space Model
$x_1 \sim \mu_0$
$x_{t+1}|x_t \sim f(x_t, u_t, t)$
$y_t|x_t \sim g(x_t, \theta_t)$
Linear/Gaussian
Noninear/Non-Gaussian
Kalman Filter Model
$x_1 \sim \mathcal{N}(\mu_{x_1}, \Sigma_{x_1})$
$x_{t+1}|x_t \sim \mathcal{N}(F_t x_t + B_t u_t, Q_t)$
$y_t|x_t \sim \mathcal{N}(H_t x_t, R_t)$
Particle Filter
Particle set
$\{\tilde{x}_i, w_i\}, i = 1, ..., N$

Figure 4.1: A subset of Dynamic Bayesian Networks

Figure 4.1.

4.2 Dynamic Bayesian Networks for Data-Driven CPS Design

In this section, we will focus on a subset of DBNs that are especially relevant to modeling data streams in the context of CPS. A large subset of applications on sensor data focus on *measuring* a quantity that is either intrinsic to a system, or is related to the environment in which the system operates. Naturally, it is a common scenario to assume an underlying dynamic system. The system would often be equipped with sensors, delivering data regarding to certain quantities within the operating system. Consider, for example, a terrestrial mobile robot, operating in an indoor environment. The robot will be subject to a dynamics model,

4.2.1 State-Space Model (SSM)

We have discussed in the previous section that the expressive power of DBNs in modeling system dynamics and establishing relationships between quantities in system design is based on the explicit modeling of *state*. A state-space model is the most general form of a DBN. This model considers a system state, which is observable via a set of noisy observations. State dynamics are modeled to be first order Markov, and a prior distribution on state is defined.

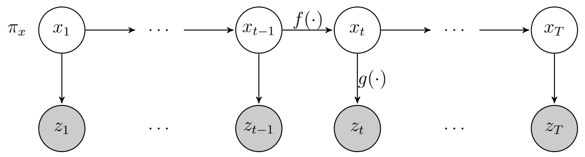


Figure 4.2: Graphical model representation of an SSM

A formal representation of a state-space model is given by

$$x_1 \sim \pi_X(x_1) \tag{4.2}$$
$$x_{t+1}|x_t \sim f(x_t, u_t, t) \tag{4.3}$$
$$\mathbf{z_t}|x_t \sim g(x_t, u_t, t) \tag{4.4}$$

where $x_t \in \mathbb{R}^K$ corresponds to the system state, $\mathbf{z_t} \in \mathbb{R}^{\mathbf{L}}$ is an L-dimensional observation of the K-dimensional state vector x, both collected at time t. $\pi_X(\cdot)$ is the prior distribution over state x, $g(\cdot)$ expresses the stochastic *measurement model* as a function of state x_t, control inputs $u_t \in \mathcal{U}$, where $\mathcal{U}$ is the domain of the control input. $f(\cdot)$ is the random function specifying the discrete state dynamics. We use the symbol $\sim$ to denote a "distributed according to" relation. A graphical model representation of a generic SSM is shown in Figure 4.2.

The SSM structure is encountered in many CPS applications, where it is of interest to track and control the physical state of a plant, however, the state is only observable via noisy measurements, that can often be modeled accurately as a parametric function of state with additive noise. One of the most widely used subset of SSMs are Kalman Filter models, which assume linear-Gaussian dynamics for $\pi(\cdot)$, $f(\cdot)$, $g(\cdot)$. In the following chapters, we will focus on the more general definition of an SSM, which encapsulates the subclass of well-known Kalman Filter models.

4.2.2 Hidden Markov Model (HMM)

A special case of SSMs, where the hidden state x takes values in a discrete set, is known as a Hidden Markov Model (HMM). Formally, an HMM is given by

$$x_1 \sim \pi_X(x_1) \tag{4.6}$$
$$x_{t+1}|x_t \sim A_{x_t,x_{t+1}} \tag{4.7}$$
$$\mathbf{z_t}|x_t \sim g(x_t) \tag{4.8}$$

where $x_t \in \{1, \ldots, K\}$ corresponds to the discrete hidden system state, A is a $K \times K$ state transition matrix, where $A_{i,j} \triangleq P(x_{t+1} = j|x_t = i)$, and $g(\cdot) \triangleq P(z_t|x_t = i)$

is the so-called emission distribution, that can either be a continuous or a discrete distribution conditioned on the true state. Finally, $\pi_X(\cdot)$ denotes the multinomial prior distribution over the state space. A graphical model representation of a generic HMM is shown in Figure 4.3.

The discretization of state space poses a fundamental difference towards the problems that can be modeled and solved by HMMs. Essentially, HMMs are very successful in modeling *model* system behavior, where each different mode of the system *emits* observations with distinct probabilistic properties.

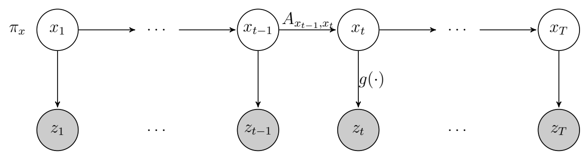


Figure 4.3: Graphical model representation of an HMM

4.2.3 Explicit-Duration Hidden Markov Model (EDHMM)

In data-driven modeling of systems, it is essential that the learning model captures relevant properties of the underlying system based on observed data. For probabilistic inference in dynamical systems whose state is only observable via state-dependent data, HMMs have been a widely used tool. However, in many CPS applications, especially for those of interest to human-in-the-loop systems, explicitly modeling event durations gain importance. In such a setting, simple HMMs often become too simplistic as they do not allow explicit parameterization of state durations, but only probabilities of one-step state transitions. An extension of HMMs that can significantly improve model fidelity for duration modeling are Explicit-Duration Hidden Markov Models (EDHMMs) [98]. These models, in addition to modeling the hidden state space as a Markov chain, also introduce the duration spent within each state as an additional hidden variable of the dynamic Bayesian network. The graphical model representation of a generic EDHMM is shown in Figure 4.4.

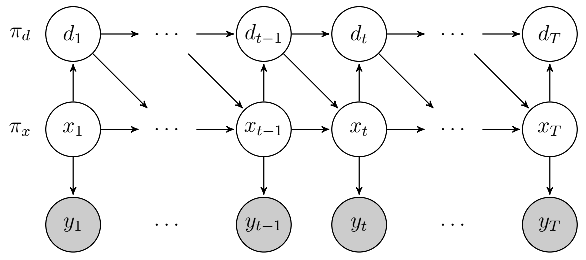


Figure 4.4: Graphical model representation of an EDHMM

The standard definition of an EDHMM models hidden state and its duration to be discrete hidden variables. The state dependent observations can be drawn from either a discrete or continuous emission distribution.

4.2.4 Common Tasks on DBNs

4.2.4.1 Inference

In relation to modeling system dynamics, perhaps the most relevant inference problem on DBNs is the so-called *filtering* or *state estimation* problem. Here, the goal is to infer x_t, the system state at time t, given noisy observations collected over a window in time. Formally, the goal is to compute the posterior probability $P(x_t|z_{1:t})$ given all observations made up to time t. Here, $z_{1:t} \triangleq \{z_1, \ldots, z_t\}$. Note that based on the SSM given by (4.2)-(4.4), the complete probability distribution of the SSM up to time t can be factored out as

$$
\begin{aligned}
p(x_{1:t}, z_{1:t}) &= p(x_1) \prod_{i=1}^{t} p(z_t|x_t) \prod_{i=2}^{t} p(x_t|x_{t-1}) && (4.9)\\
&= \pi_X \prod_{i=1}^{t} g(x_t) \prod_{i=2}^{t} f(x_{t-1})\,. && (4.10)
\end{aligned}
$$

This form of the probabilistic model proves useful in many inference applications. Specifically, it is possible to compute the state estimate x_t by using the recursive relation

$$
\begin{aligned}
p(x_t, z_{1:t}) &\triangleq \alpha(x_t)\\
&= \int_{x_{t-1}} p(x_t, x_{t-1}, z_{1:t}) d_{x_{t-1}}\\
&= \int_{x_{t-1}} p(z_t|x_t, x_{t-1}, z_{1:t-1}) p(x_t|x_{t-1}, z_{1:t-1}) p(x_{t-1}|z_{1:t-1}) d_{x_{t-1}}\\
&= \int_{x_{t-1}} p(z_t|x_t) p(x_t|x_{t-1}) p(x_{t-1}|z_{1:t-1}) d_{x_{t-1}}\\
&= g(x_t) \int_{x_{t-1}} f(x_{t-1}) \alpha(x_{t-1}) d_{x_{t-1}}\,,
\end{aligned}
$$

from which the estimate $p(x_t|z_{1:t})$ simply follows by normalizing this recursive quantity by $p(z_{1:t})$.

For a subset of the SSMs, in which $f(\cdot)$ and $g(\cdot)$ are linear functions with Gaussian noise assumptions, the estimation problem can be delegated to a Kalman filter, which yields the optimal solution in the minimum mean-square error sense [40]. Variants of the Kalman filter for nonlinear state transition and measurement models exist, although these methods rely on local linearizations of the nonlinear quantities, as in the Extended Kalman Filter (EKF), or on sampling techniques such as in the Unscented Kalman Filter (UKF). Although these variants enable efficient computation, the need for linear approximations of measurement models, Gaussian assumption, and a requirement for a point guess for the initial state severely limit Kalman Filter models' advantages for a general SSM. Besides the suboptimality of results, these methods also suffer from underestimated variance, bias, and consequent deviation from the true state estimate [95, 99].

A more general Bayesian filtering approach, known as particle filtering, fits to a much wider set of noise and measurement models, and outperforms Kalman Filter variants in handling multi-modal distributions.

4.2.4.2 Particle Filtering for State Estimation

Particle filtering is a sequential Monte Carlo method that approximates the posterior distribution $p(x_t|\mathbf{z}_{1:t})$ with a weighted particle set, where each *particle* is a candidate state estimate, its weight being proportional to its likelihood. The filter approximates the posterior state with a probability mass function estimate, $\hat{p}(\cdot)$, given by

$$\hat{p}(x_t) = p(x_t|\mathbf{z}_{1:t}) = \sum_{i=1}^{N} w_t^i \delta(x_t - \tilde{x}_t^i) \tag{4.11}$$

where $\tilde{x}_t^i$ denotes the i'th particle for the state estimate at time t, $\delta(\cdot)$ is the Dirac delta function and w_t^i is the weight associated with the particle i such that $\sum_{i=1}^{N} w_t^i = 1$, where N is the total number of particles. This approximate probability mass can be used to approximate the MMSE estimate of x_t as

$$\hat{x}_t^{MMSE} = \mathbb{E}[x_t] \approx \sum_{i=1}^{N} w_t^i \tilde{x}_t^i \tag{4.12}$$

This provides a state estimate at time t as a weighted mean of the set of particles present at this time step. At this point, it is important to highlight that the choice of a particle filter over parametric estimation methods like Kalman filters in actor-oriented settings is that downstream actors have direct access to particles, and would have direct control over tuning the subset of particles used in approximate algorithms. The ability to do so is important in a ubiquitous computing scenario, for which varying computational resources may require on-line algorithms to adapt their resources to compromise accuracy for better real-time performance.

Furthermore, the particle representation can be used to approximate the marginal measurement density by

$$
\begin{aligned}
p(\mathbf{z}_t) &= \int_X p(\mathbf{z_t}|x_t)p(x_t)dX && (4.13)\\
&\approx \int_X p(\mathbf{z_t}|x_t)\hat{p}(x_t)dX && (4.14)\\
&= \sum_{i=1}^{N} w_t^i p(\mathbf{z_t}|\tilde{x}_t^i) && (4.15)
\end{aligned}
$$

where we have replaced $p(x_t)$ with $\hat{p}(x_t)$, which is given by (4.11).

The particle filter formulations that are of interest to the tools discussed in this thesis have been presented in Appendix A. A more comprehensive formulation of sequential Monte Carlo methods along with specific particle filtering algorithms are presented in [40].

4.2.4.3 Parameter Estimation (Model Learning)

The previously introduced problem of inference relies on the assumption that a SSM is already at hand, i.e., the model parameters are known. A dual problem in the domain of DBNs is to estimate model parameters, given a sequence of observations. Specifically, the problem is to find the optimal parameter set λ^* such that

$$\lambda^* = \arg\max_{\lambda} p(y_1, \ldots, y_T \mid \lambda)\,.$$

Here, optimality criterion is defined in the maximum-likelihood sense.

4.2.4.4 Classification

A common problem, that applies to DBNs with a discrete state space is known as the classification problem. This is often analogous to the state inference problem explained for continuous state-space models. Formally, the goal is to find the most likely sequence of true state, given observations up to time t:

$$C^*(y_t) = \arg\max_{C} \quad p(y_{1:t}|C)P(C)\,, \tag{4.16}$$

where $p(y_{1:t}|C)$ is the likelihood of the model considering a certain class sequence $C := \{c_1, \ldots, c_t\}, c_i \in X$, and $P(C)$ are the class priors.

Chapter 5

Modular Design of Data-Driven CPS

In the previous chapters, the need for modeling formalisms that enable modular development of complex CPSs and introduced aspect-oriented modeling as a tool for handling cross-cutting concerns in complex CPS have been discussed. Then, Dynamic Bayesian Networks were introduced as a well-suited family of algorithms to infer and learn controllable quantities in system design, given streaming data. An outstanding concern is to define the *interface* between inference and learning algorithms, and the system model itself. While the methodology will be widely applicable across domains, we will adapt the domain of cooperative mobile robot control throughout the chapter to demonstrate the applied value of the defined interfaces.

Many cooperative multi-robot applications require the co-design of dynamics, control, and data-driven machine learning techniques to enable adaptive and resilient workflows that can adapt to environmental constraints. One such application has been described in section 3.4, where have discussed the use of a simple communication aspect for better encapsulation of implementation details of inter-robot communication. In industrial-scale applications, the extent of cross-cutting concerns in such complex systems go far beyond communication models. Consider, for instance, a target tracking robot, whose purpose is to estimate the *state* of a target in an indoors environment. Here, the target state is often defined to include a position in Euclidean space, as well as the target's velocity and acceleration in this space. We have discussed in chapter 4.2.1 that state-space models are especially useful for representing dynamical systems and estimation problems for these systems. The search robot would typically be equipped with sensors such as laser range-finders, cameras, and other wireless sensors such as BLE transmitters, that let it map its environment and the target in it. The heterogeneity of the sensors that enable robot perception is in fact very valuable for the purposes of localization and mapping tasks, however, they also introduce many challenges in *correctly* interpreting the streaming data received from each source. In many applications, it is not uncommon for the statistics of data received from a sensor to significantly vary as a function of its environment [32]. Such variation may be due to factors such as line-of-sight between sensor and target, material composition of the environment, as well as wireless interference. This would require the system model to interpret data differently according to the estimated environmental conditions, in order to converge to a meaningful estimate of the target state. For even the seemingly simple task of interpreting sensor data for target state estimation, it becomes essential for inference algorithms to receive faithful measurement models, as well as estimated environmental conditions.

The need for defining interfaces between learning algorithms and state-space models is not limited to state estimation applications alone. In fact, they are also essential for control tasks, where awareness of the robot dynamics, as well as interpreting environmental conditions from streaming data becomes key to enabling safe and optimal data-driven control. Considering the increasing need for an interface between system models and advanced learning and optimization algorithms, we will next discuss implementation of such interfaces in the form of an actor-oriented toolkit, which will build upon the aforementioned key enablers of Aspect-Oriented Modeling and DBNs.

5.1 Motivation

A fundamental challenge in distributed CPS design is the gap between computation requirements and existing design tools for adaptive real-time simulation and deployment. Numerous closed-loop CPS applications depend on on-line optimization of quantities that may include estimation error, trajectories, and information theoretic quantities such as entropy. Although there exists a wealth of tools to build models and perform machine learning tasks on the data, most of these tools are designed for batch processing and off-line data classification. Closed-loop data intensive CPS applications bring about unique requirements for such tools:

- Unlike off-line classification, estimation, and inference problems, swarm applications are subject to variable availability of sensor data and network resources, due to the cyber-physical setting they operate in. Effects such as imperfect communication channels, anomalies, and latency, which may also require dynamic adaptations of learning, models must be considered.

- Sensor network applications often rely on optimization of quantities that are either non-convex, hard to express analytically, or intractable in closed form. This renders data-driven approximations of these cost functions desirable in being tractable and fast.

- In swarm applications, especially for a ubiquitous computing scenario, available computation resources may widely vary. This creates a demand for possibly adaptive approximations to optimization problems, in which algorithmic accuracy can be traded off for better real-time performance, without requiring a reprogramming of the sensor network.

These novel challenges introduced by the large scale design and control of distributed CPS suggests that a shift in the formalisms used traditionally to design dynamics, coordination, and control aspects of theses systems can tremendously benefit the programmability and interpretability of systems.

5.1.1 Related Work

Recent advancements in cooperative multi-agent applications have enabled advanced robotic hardware to be combined with efficient control algorithms to perform safety-critical tasks using cooperative robot teams in real-time [61, 33, 42, 52, 103]. As these systems get more complex, especially in mobile scenarios that require a number of independent robots to cooperate towards a common goal, reusable modules for managing dynamics and control becomes an immediate necessity. Moreover, since most robot hardware is custom-built, engineers often need to develop custom software to enable hardware interaction, which ends up being time consuming and error prone. Therefore, libraries that create an abstraction between hardware and software become essential for modularity. The Robotic Operating System (ROS) [97], an open-source framework that aims at providing reusable robot code for mediating communication, perception and control tasks of generic robotic systems, has been tremendously successful and has become a universal platform for robotic development.

ROS owes its success to the goal of rapid prototyping of reliable heterogeneous robotic systems and enabling researchers to focus more on novel aspects of robotics challenges including adaptive, data-driven, resilient applications without repeating infrastructure, monitoring, and visualization work. In the same vein, other frameworks for rapid prototyping of specialized platforms that leverage development of pattern recognition tasks on interactive devices have also been proposed [59, 84]. One other related framework in this area is presented in [28], which introduces a high-level programming language for robotic applications based on ROS.

Most of these frameworks rely on libraries based on imperative code for unifying control algorithms, robot kinematics, sensor models and environmental constraints. Because of this, the limitations of monolithic code in representing cross cutting concerns and capturing timing persist in resulting frameworks.

The multidisciplinary field of CPS design has triggered intensive research on programming foundations of distributed heterogeneous systems that focus on eliminating these fundamental challenges in modeling [38, 77]. Researchers have investigated a collection of programming paradigms to be able to correctly express CPS behavior and be able to perform model-based design, synthesis, and verification. Frameworks such as Simulink, LabVIEW and Ptolemy II [96] have adopted actor-oriented design methodologies and are widely used for design of embedded systems. Actor-oriented design in Ptolemy II has been studied in the context of several CPS domains including smart grid, aircraft, and automotive systems [38].

Although these actor-oriented tools present general-purpose frameworks for heterogeneous design of CPS, some already offer domain-specific toolkits. For example, MATLAB provides a Robotics System Toolbox for developing autonomous mobile robotics applications [13] that also supports ROS integration. Adapting a similar approach, we will develop a domain-specific aspect-oriented library that builds upon the Ptolemy II framework. The goal is to benefit from the available programming foundations presented by Ptolemy II to build correct-by-construction data-driven programs. We will also explain how the presented framework will be integrable with mature tools such as ROS.

5.2 PILOT: An Actor-oriented Learning and Optimization Toolkit

PILOT (Ptolemy Inference, Learning, and Optimization Toolkit) is a library of aspect-oriented components that is built upon the Ptolemy II core library [20]. The main purpose of PILOT is to enable Bayesian inference, actor-oriented optimization, and state-space modeling in actor models. The precise implementation is based on the philosophy that implementation, environment and hardware-specific concerns of CPSs should be *aspects* of a system design. Moreover, the meaning and operation of specific learning and inference tasks should be configurable by these aspects. PILOT utilizes the aspect mechanism presented in Chapter 3 to implement state-space models and sensor models that *decorate* inference and optimization actors to give them operational meaning.

A system architecture diagram of PILOT is given by Figure 5.12, that conceptually demonstrates the role of modular state-space models as aspects in the design of estimation and control tasks. To consolidate the meaning of aspect-oriented development of these algorithms as hinted by Figure 5.13, we continue with specific aspects and their implementation in PILOT.

5.2.1 DBN-Based Parameter Estimation and Classification in PILOT

Before focusing on dynamics and sensing models in the context of cooperative mobile-robot applications, we will introduce a broad set of learning actors available as part of PILOT. Specifically, we will introduce several Dynamic Bayesian Network types as PILOT actors and aspects, and how parameter estimation and classification tasks are represented. Table 5.1 summarizes the customized DBNs that are made available as part of the PILOT library. Although those listed constitute a small subset of all available DBNs, we believe they cover the most widely-used set of DBNs for CPS applications and facilitate development of any custom DBNs if needed. Also, it is possible to obtain special cases of widely used Bayesian Networks such as Gaussian Mixture Models (GMMs) by omitting the state-transition parameters of the dynamic variants of these models.

5.2.1.1 Hidden Markov Models in PILOT

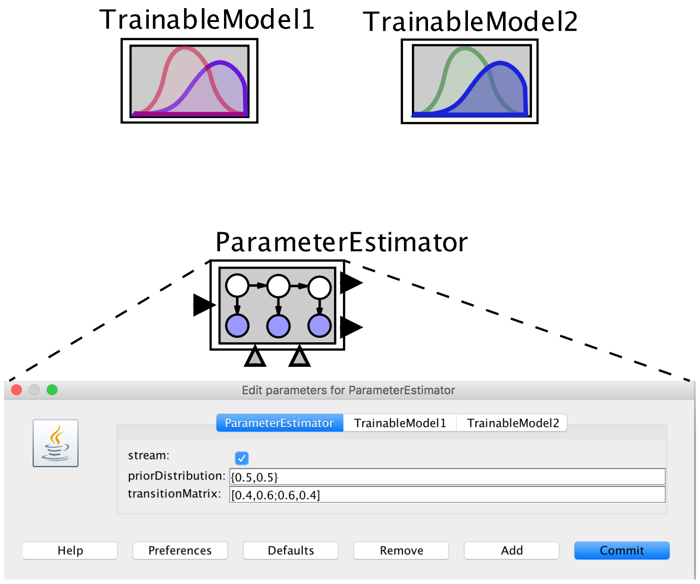
Chapter 4 provides a detailed definition of DBNs of interest to PILOT. In this section, we will demonstrate the implementation of an HMM in PILOT. We will keep the aspect associations generic, as they will apply for any other type of emission distribution and DBN that will be represented in PILOT.

An HMM is given by

$$\begin{aligned} x_1 &\sim \pi_X(x_1) \\ x_{t+1}|x_t &\sim A_{x_t,x_{t+1}} \\ \mathbf{z_t}|x_t &\sim g(x_t) \end{aligned} \tag{5.2}$$

where $x_t \in \{1,\ldots,K\}$ is the discrete state, A is a state transition matrix, where $A_{i,j} \triangleq P(x_{t+1}=j|x_t=i)$ is the state transition matrix, and $g(\cdot) \triangleq P(z_t|x_t=i)$ is the state-dependent emission distribution. PILOT offers two inference actors for tasks on HMMs, namely, a `ParameterEstimator`, and an `ObservationClassifier`. While these tasks have been formally defined in Chapter 4, the interpretation of the tasks in the context of a cyber-physical system application is considerably different. Often, for an HMM that is operating on streaming data, for instance, collected from a sensor, the state will correspond to a discrete state of operation, in which the sensor data is expected to follow a state-dependent distribution. By modeling it so, transitions of state will be visible through changes in the statistics of the sensor data.

In line with this definition, PILOT implements an aspect interface called `TrainableModel`, which specifies an abstract emission distribution that can be associated with a specific state of operation in the system model. A specific parametric emission distribution, e.g., Gaussian, Poisson, etc., implements the `TrainableModel` interface and defines distribution-specific parameters that can be learned via a `ParameterEstimator`. Conversely, given an instance of `TrainableModel` with specified parameters, an `ObservationClassifier` can refer to these parameters to classify an observation using maximum-likelihood estimation based on the emission parameters. This aspect association is demonstrated in Figure 5.1, where an HMM parameter estimator is automatically decorated by all instances of `TrainableModel` that exist within the scope of the model. If enabled, the parameters of these models will be approximated by the `ParameterEstimator`.



stream: ✓
priorDistribution: {0.5,0.5}
transitionMatrix: [0.4,0.6;0.6,0.4]

Figure 5.1: A PILOT HMM Parameter Estimator that defines Emission Distributions via Aspect Associations

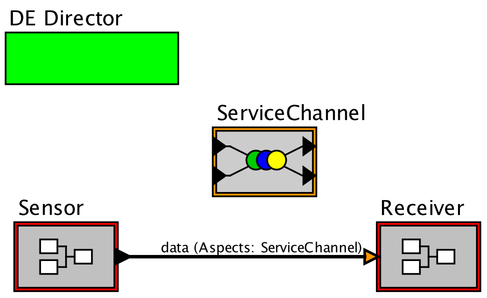


Figure 5.2: A Communication Aspect Describing a Wireless Communication Channel Between Two Devices

As a concrete example, consider a communication aspect representing the wireless communication channel between two devices, a `Sensor` and a `Receiver`, as given by Figure 5.2 (a similar scenario was also discussed in Section 3.4). Here, we will assume the common case of multiple *states* of communication to be available for transmitting each packet between the source and the destination, each having a distinct latency profile. For simplicity, assume that the available means of communication are a reliable WiFi network, a reliable LTE network, and a collection of unreliable higher latency communication sources. It is a common scenario for mobile devices to switch their modes of communication on a per-packet bases, given a multitude of available networks for transmitting information. We will also assume periodic data transmission where one packet is transmitted at each time step.

We are interested in designing an anomaly detection application at the `Receiver` node given in Figure 5.2. Assume that the receiver node would like to *infer* which means of communications was used to transfer each packet that has been delivered to this node, and if many packets are thought to be transmitted through a high latency link, an alert should be triggered. Omitting any clock drift between the `Sensor` and the `Receiver`, the communication latency can be measured at the receiver node, and based on the latency statistics of communication channels, one can classify each latency observation to one of the available states of communication. A Hidden Markov Model would be a good fit for such a problem setting, where the observed data sequence is the packet transmission latency for each packet being delivered, the hidden state is the actual means of communication used for the transmission of each packet, and the switching between the networks is following a Markov process. Such a model assumes the form in (5.2), with $X := \{S_0, S_1, S_2\}$, where S_i are the WiFi, LTE, and high-latency network states respectively, and $g(S_i) = p(z_t|x_t = S_i) \sim Rice(\nu_i, \sigma_i) \approx \mathcal{N}(\mu_i, \sigma_i)$ is the latency distribution of each communication state. The state transition matrix A is a 3×3 matrix, and π_X is a 3×1 prior vector.

HMMs as Generative Models in PILOT: An HMM can be defined as a generative model in Ptolemy using modal models [96]. Here, the finite state machine given by Figure 5.3 represents the *hidden* states of the HMM, and the emissions are implemented as *state refinements* of the modal model. The state transition probabilities are implemented via a built-in function $probability(\cdot)$ for state-machines that has been implemented as part of PILOT (see the implementation of the FSMActor class for details). In this example, each transition label defines a transition that is taken with probability $p(x_{t+1} = j|x_t = i) := A_{i,j}$. By simulating this state machine for multiple iterations, a histogram of latencies can be obtained as given by Figure 5.4. Here, each sample is labeled by the hidden state it was generated by for demonstration purposes. In a real application, this label is considered to be an unobserved variable.

Such generation capabilities is often useful for emulating a multi-modal stochastic system, where it is of interest to simulate the behavior of a dynamic Bayesian network in advance and model a system based on the simulation, eventually replacing the simulated data source with the real data streams, and expecting the behavior to remain faithful to the modeled one.

Parameter Estimation and Classification on HMMs:

Given an HMM model of the communication link, one task would be *training* the parameters of the model to best represent the actual behavior of the system. Here, the parameters of the model to be trained are $\lambda = \{\pi_X, A, \{\mu_i, \sigma_i\}\}$. The second, and perhaps main task to be defined is the actual anomaly detection task, carried out via *classifying* incoming data points into the most likely state of communication, where

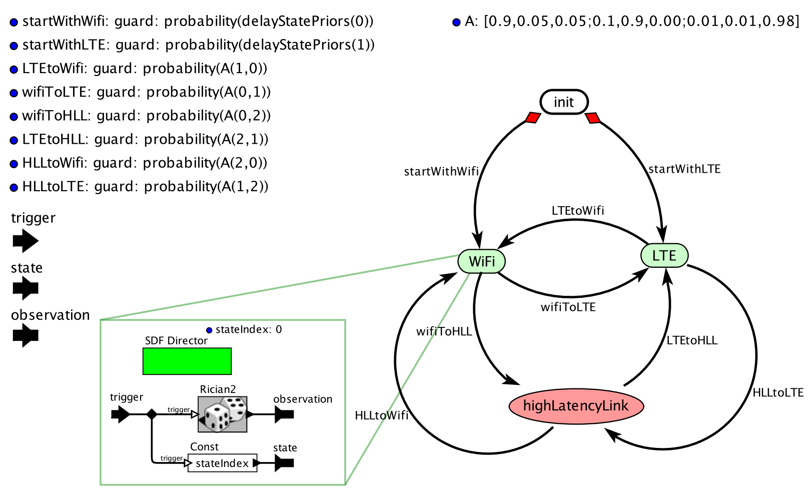

startWithWifi: guard: probability(delayStatePriors(0))
startWithLTE: guard: probability(delayStatePriors(1))
LTEtoWifi: guard: probability(A(1,0))
wifiToLTE: guard: probability(A(0,1))
wifiToHLL: guard: probability(A(0,2))
LTEtoHLL: guard: probability(A(2,1))
HLLtoWifi: guard: probability(A(2,0))
HLLtoLTE: guard: probability(A(1,2))
A: [0.9,0.05,0.05;0.1,0.9,0.00;0.01,0.01,0.98]
trigger
state
observation
init
startWithWifi
startWithLTE
LTEtoWifi
WiFi
LTE
wifiToLTE
wifiToHLL
LTEtoHLL
HLLtoWifi
HLLtoLTE
highLatencyLink
stateIndex: 0
SDF Director
Rician2
trigger
observation
Const
stateIndex
state

Figure 5.3: PILOT implementation of the HMM representing the three-state communication link

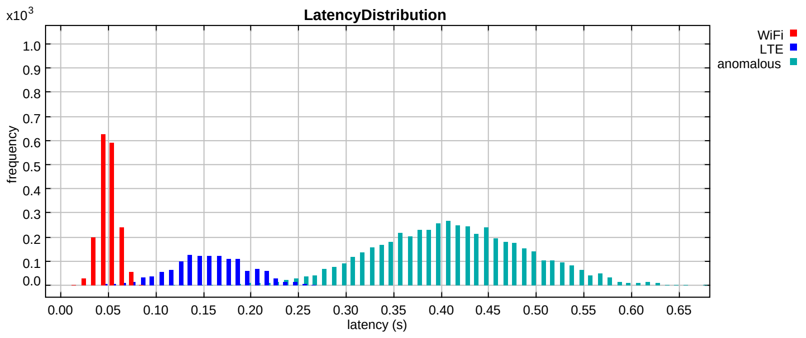


Figure 5.4: Simulated Latency Histograms per Channel

classifications into the *highLatencyLink state* (S_2) would be considered anomalies.

Figure 5.6 depicts two PILOT actors for parameter estimation and observation classification on HMMs, respectively. Note that the emission distribution parameters $\{\mu_i, \sigma_i\}$ are contained in the `TrainableGaussianModel` aspect for each state, and can be altered by the `HMMGaussianEstimator` as the model is trained given input data. Even though the trainable aspects are kept abstract for the sake of this example, they would often refer to aspects that indeed affect other parts of the system design, therefore, updating model parameters via training would automatically update the effect of these aspects in the different parts of the design. This propagation will be exemplified further in the upcoming sections, particularly in Chapter 7.

Finally, the `HMMGaussianClassifier` actor classifies each incoming latency observation into either one of the aspects it is decorated by. The classification labels reference the classified state by name. Figure 5.5 displays an example classification trace of true vs. classified state of latency samples, obtained by simulating this model.

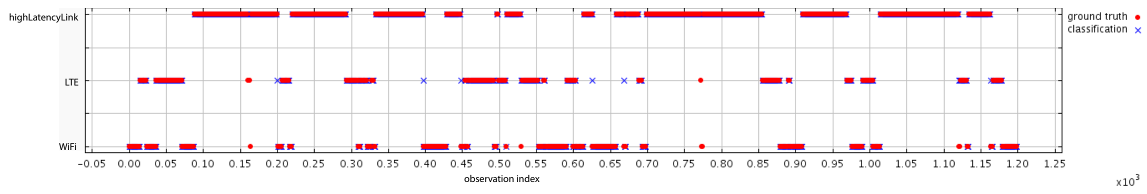


Figure 5.5: True vs Classified State Sequences for the Communication Anomaly Detection Classification Example

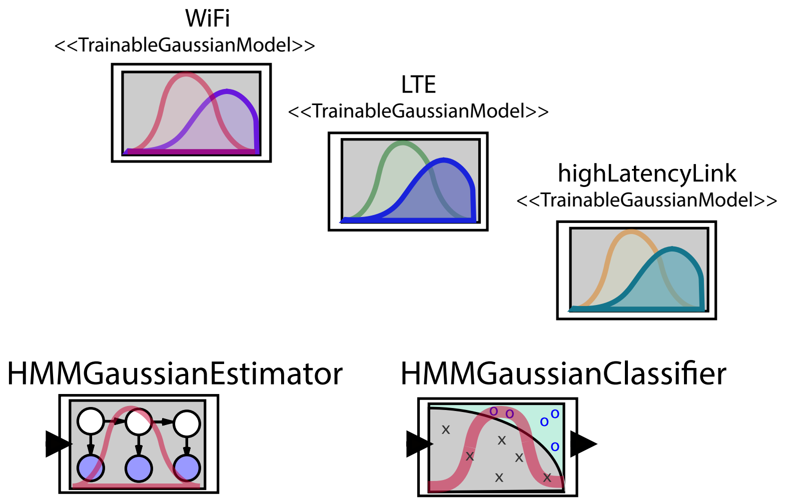

WiFi
<<TrainableGaussianModel>>
LTE
<<TrainableGaussianModel>>
highLatencyLink
<<TrainableGaussianModel>>
HMMGaussianEstimator
HMMGaussianClassifier

Figure 5.6: Aspect-Oriented Parameter Estimation and Classification on HMMs

5.2.2 Aspect-Oriented Sensor Models and Robot Dynamics

In industrial settings, mathematical models of component dynamics, kinematics, as well as models of uncertainty and noise are key to building complex systems. Moreover, these mathematical models are often shared between different pieces of the functional model. In a cooperative robotics scenario, robot dynamics are often given by a state-space model (SSM), and is used to model the behavior of robots in a physical environment. Also, noise characteristics of on-board sensors are required for accurate quantification of the information content of sensor measurements. Clearly, these dynamics models need to interact with one another for cooperative inference and control algorithms.

Model	Supported Emission Distributions	Aspect Association
Hidden Markov Model (HMM)	Gaussian Exponential Multinomial Poisson	TrainableModel*
Hidden Semi-Markov Model (HSMM)	Gaussian Exponential Multinomial Poisson	TrainableModel*

* indicates that actor supports multi-aspect association

Table 5.1: Summary of PILOT Bayesian Network Library Components

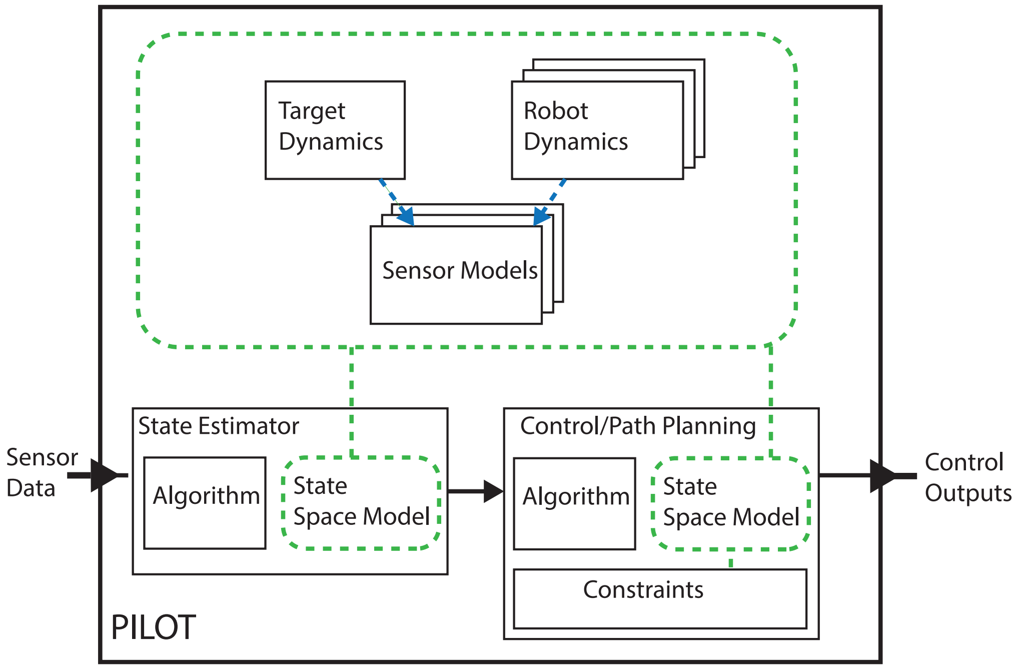


Figure 5.7: State-Space Aware System Architecture in PILOT

For maintaining modularity in design, the SSM can be refactored into two parts: *dynamics* and *sensing*. State-Space Dynamics represent motion in a state-space, which often has deterministic and stochastic elements to it. The deterministic model often explains a motion model for a physical component, which is affected by process noise. Also, the dynamics model can include an initial condition, which can be given as a deterministic state in the state-space, or as a prior distribution over it. Formally, the state-space dynamics are given by

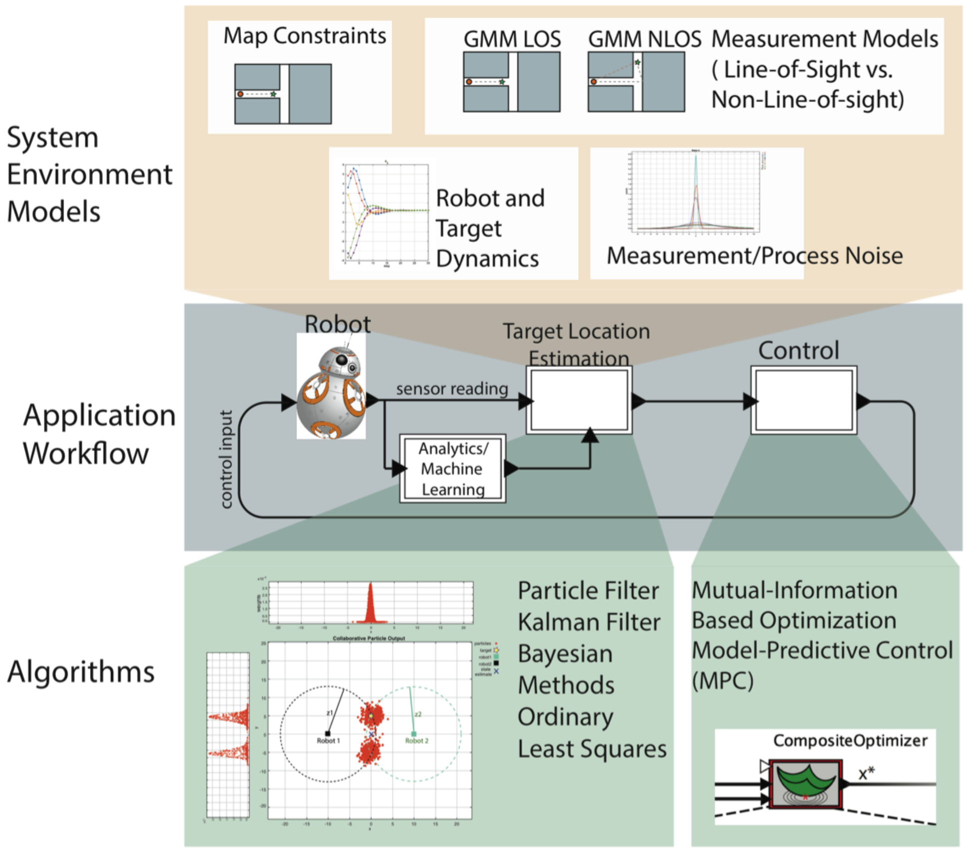
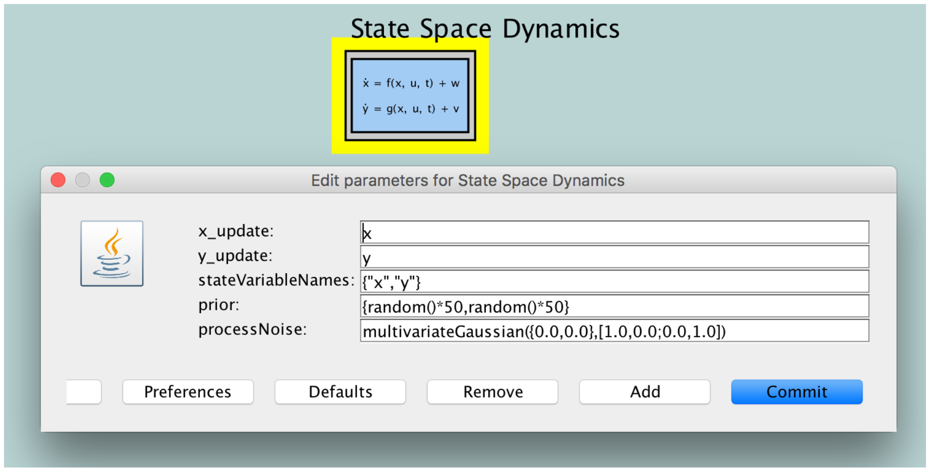

System Environment Models
Map Constraints
GMM LOS
GMM NLOS
Measurement Models (Line-of-Sight vs. Non-Line-of-sight)
Robot and Target Dynamics
Measurement/Process Noise
Application Workflow
Robot
sensor reading
Target Location Estimation
Control
Analytics/ Machine Learning
control input
Algorithms
Particle Filter
Kalman Filter
Bayesian Methods
Ordinary Least Squares
Mutual-Information Based Optimization
Model-Predictive Control (MPC)
CompositeOptimizer
x*

Figure 5.8: Aspect-Oriented Application Design in PILOT

$$x_1 \sim \pi_X(x_1) \tag{5.4}$$
$$x_{t+1}|x_t \sim f(x_t, u_t, t)\,, \tag{5.5}$$

where $x_t \in \mathbb{R}^K$ corresponds to the system state, $\pi_X(\cdot)$ is the prior distribution over state, and $f(\cdot)$ is the random function specifying the discrete state dynamics. The dynamics is often affected by the control inputs $u_t \in \mathcal{U}$, where $\mathcal{U}$ is the domain of the control. We use the symbol $\sim$ to denote a "distributed according to" relation.

The aspect representation of the state-space dynamics encapsulates (5.4)-(5.5) as an aspect that can give semantic meaning to downstream inference actors that depend on a state-space dynamics to function. Figure 5.9 illustrates the PILOT implementation of a state-space dynamics model, which has the following state-space



x_update:	x
y_update:	y
stateVariableNames:	{"x","y"}
prior:	{random()*50,random()*50}
processNoise:	multivariateGaussian({0.0,0.0},[1.0,0.0;0.0,1.0])

Figure 5.9: The `State Space Dynamics` Aspect

model:

$$x = \begin{bmatrix} x_{\mathrm{x}} \\ x_{\mathrm{y}} \end{bmatrix} \tag{5.6}$$

$$x_1 \sim \mathcal{U}([-25, 25], [-25, 25]) \tag{5.8}$$

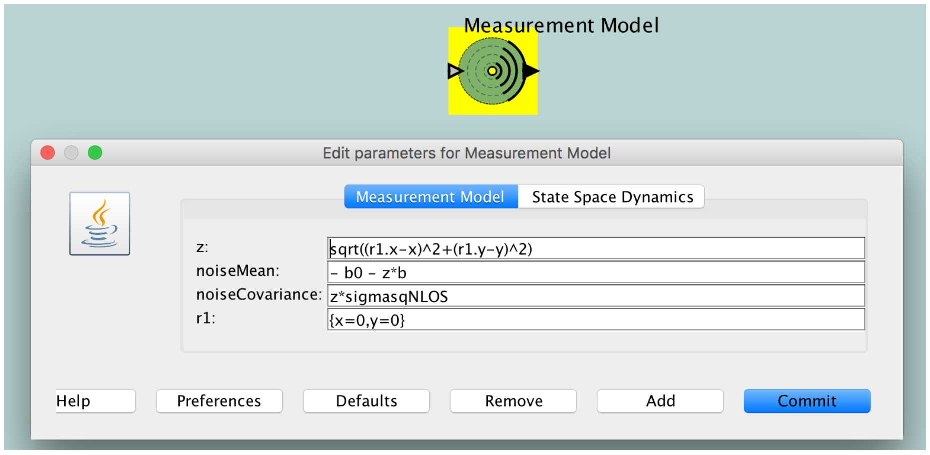
$$x_{t+1}|x_t = x_t + \nu_t, \quad \nu_t \sim \mathcal{N}(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}) . \tag{5.9}$$

Here, the represented *state* is a point in 2-D Euclidean space with `x` and `y` components, represented by x_{x} and x_{y} respectively. The prior state is unknown, and has a uniform distribution in a 50×50 square, centered at the axis origin. The state dynamics is given by a random walk, where process noise is distributed according to a standard bivariate Gaussian.

The second component of the SSM is the measurement model, which describes how the state is perceived, often via noisy sensors. If the state is observed by more than one sensor as in the example in the introduction, the measurement model can include an array of models, each for one physical sensor. Similar to the dynamics, the measurement model often consists of a deterministic description of the measurement as a function of state, as well as stochastic noise. Formally, the measurement model is described by

$$\mathbf{z_t}|x_t \sim g(x_t, u_t, t) , \tag{5.10}$$

where $\mathbf{z_t} \in \mathbb{R}^{\mathbf{L}}$ is an L-dimensional observation of the K-dimensional state vector x, and $g(\cdot)$ expresses the stochastic *measurement model* as a function of state x_t. A measurement model aspect in PILOT is given by Figure 5.10, which implements



z: sqrt((r1.x-x)^2+(r1.y-y)^2)
noiseMean: - b0 - z*b
noiseCovariance: z*sigmasqNLOS
r1: {x=0,y=0}

Figure 5.10: The `Measurement Model` Aspect

a Line-of-Sight (LOS) range measurement of the state, which typically follows an analytical model given by

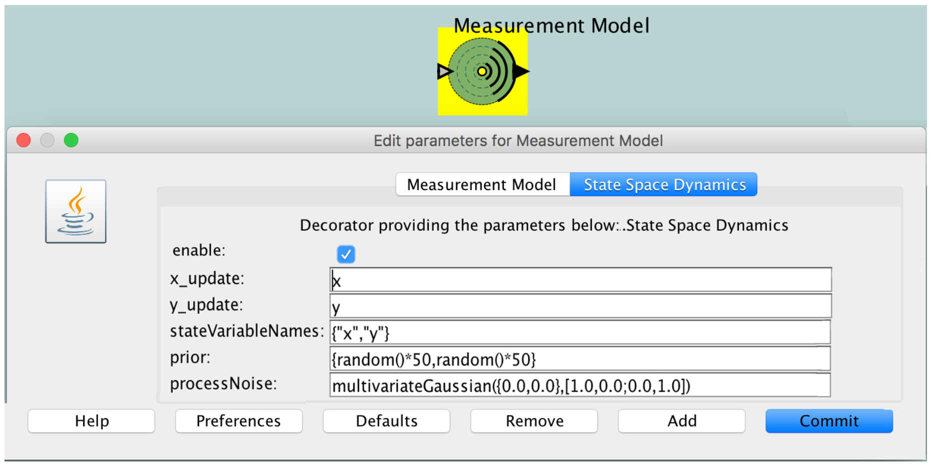
$$z_t | x_t = r + \omega_t\,, \tag{5.11}$$

$$r \triangleq \|x_t - R_t\|, \tag{5.12}$$

$$\omega_t \sim \mathcal{N}(-b_0 - br, r * \sigma_r^2)\,. \tag{5.13}$$

Here, r represents the true range between two points in 2-D space, x_t being the controlled state, and R_t being an uncontrolled state. The measurement noise is a Gaussian with a covariance that increases linearly with true distance. The measurement noise also has a positive bias that increases linearly with true distance ($b_0, b < 0$, and has an offset (The positive measurement bias in LOS conditions arise from the time difference of arrival (TDoA) approximation of the true distance, which tends to over-estimate the distance when no LOS is present[32]).

Note that the controlled state x that is being observed by the sensor, whose measurements are given by the measurement model in Figure 5.10 is not re-defined as part of the measurement. Instead, the `Measurement Model` aspect is *decorated* by the `State-Space Dynamics` aspect, which provides operational meaning to the measurement. This association is specified as can be seen by a tab in the user interface. The `State-Space Dynamics` aspect automatically decorates the Measurement Model, according to rules specified by the PILOT engine, and is *enabled* by the designer as shown in the view illustrated by Figure 5.11. Notice that a change in the state-space model and dynamics thereby automatically affects how the measurement model utilizes state. This principal will apply to all aspect-oriented actors developed by PILOT and is the core of the modular representation of cross-cutting concerns.



Decorator providing the parameters below:.State Space Dynamics

enable:	✓
x_update:	x
y_update:	y
stateVariableNames:	{"x","y"}
prior:	{random()*50,random()*50}
processNoise:	multivariateGaussian({0.0,0.0},[1.0,0.0;0.0,1.0])

Figure 5.11: Aspect Association for `State Space Dynamics`

Algorithm	Aspect Association	Supported Tasks
Particle Filter	StateSpaceModel, MeasurementModel*	State Estimation
Unscented Kalman Filter (UKF)	StateSpaceModel, MeasurementModel*	State Estimation
State Space Simulator	StateSpaceModel	Simulation
State Predictor	StateSpaceModel	Prediction
Composite Optimizer	StateSpaceModel, MeasurementModel*	Convex Optimization, Gradient Descent

* indicates that actor supports multi-aspect association

Table 5.2: Summary of PILOT Aspect Library Components

5.2.3 Multi-Modal Measurement Aspects and Dynamic Model Association

The separation of state-space dynamics and measurement models enables modal representations of sensors that refer to the same underlying state-space, but exhibit different behaviors under different conditions. As a widely encountered example in robotics applications, we will demonstrate the use of this mechanism for handling conditional measurement models for range-only sensing.

Empirical and theoretical analyses support that range-only sensing models behave significantly differently under line-of-sight (LOS) and no line-of-sight (NLOS) conditions [33]. Considering an application where mobile robots detect targets using range-only sensors (e.g., as quantified by TDoA in RF signals or Received Signal Strength(RSS) in Bluetooth Low-Energy sensors), the target's location to be estimated given sensor data will depend highly on the fidelity of the measurement model. Given that the LOS condition between the sensor and the target will continuously be subject to change in a way that can't be directly anticipated or observed by the

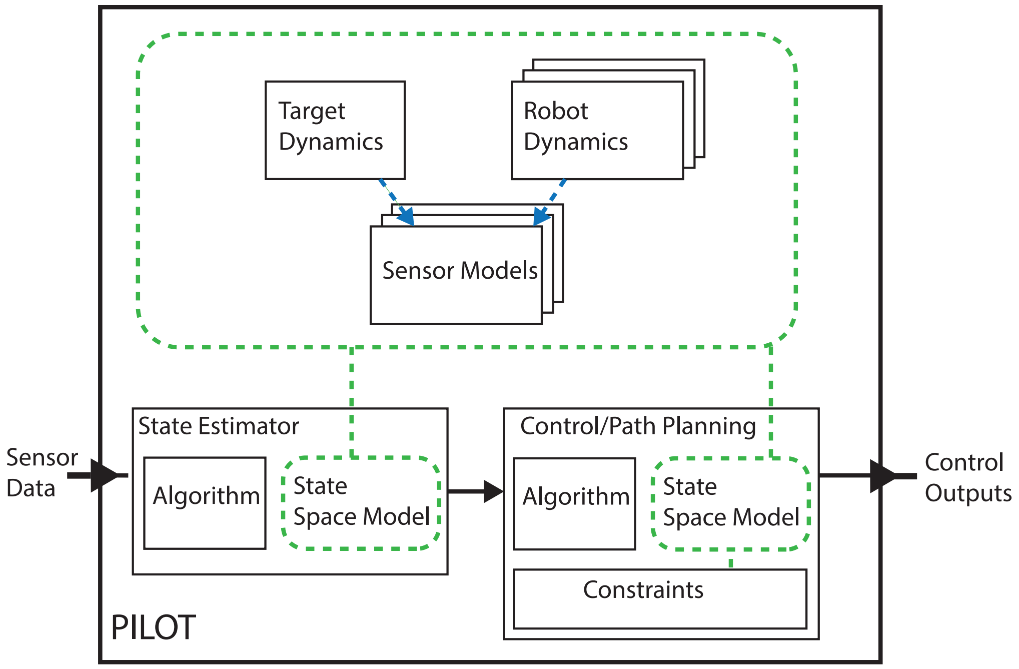


Figure 5.12: State-Space Aware System Architecture in PILOT

sensing platform, the sensing robot will need to infer the LOS condition to be able to interpret any measurement made in an accurate way.

The described problem requires one sensor model for each line-of-sight condition to be present in the system model to be able to represent the modality of the sensor models, as well as to consider the LOS condition estimate while interpreting the measurements. Both these concerns are successfully addressed by PILOT's aspect mechanism.

5.2.3.1 Representing LOS/NLOS Conditions via Measurement Aspects

Sensing applications that rely on range-only measurements of target locations tend to model LOS and NLOS measurement statistics separately as to accurately interpret observed measurements. Typically, a range measurement model is represented by

$$z_t|x_t \sim \begin{cases} \mathcal{N}(\alpha_0 + r\alpha, r\sigma^2_{LOS}) & \text{(LOS)} \\ \mathcal{N}(\beta_0 + r\beta, r\sigma^2_{NLOS}) & \text{(NLOS)} \end{cases}, \tag{5.14}$$

where r is the true distance being measured, α_0 and β_0 represent range offsets and α, β represent parameters that quantify the distance dependent noise bias of the measurement. Typically, the variance is modeled to have a linear dependence to true

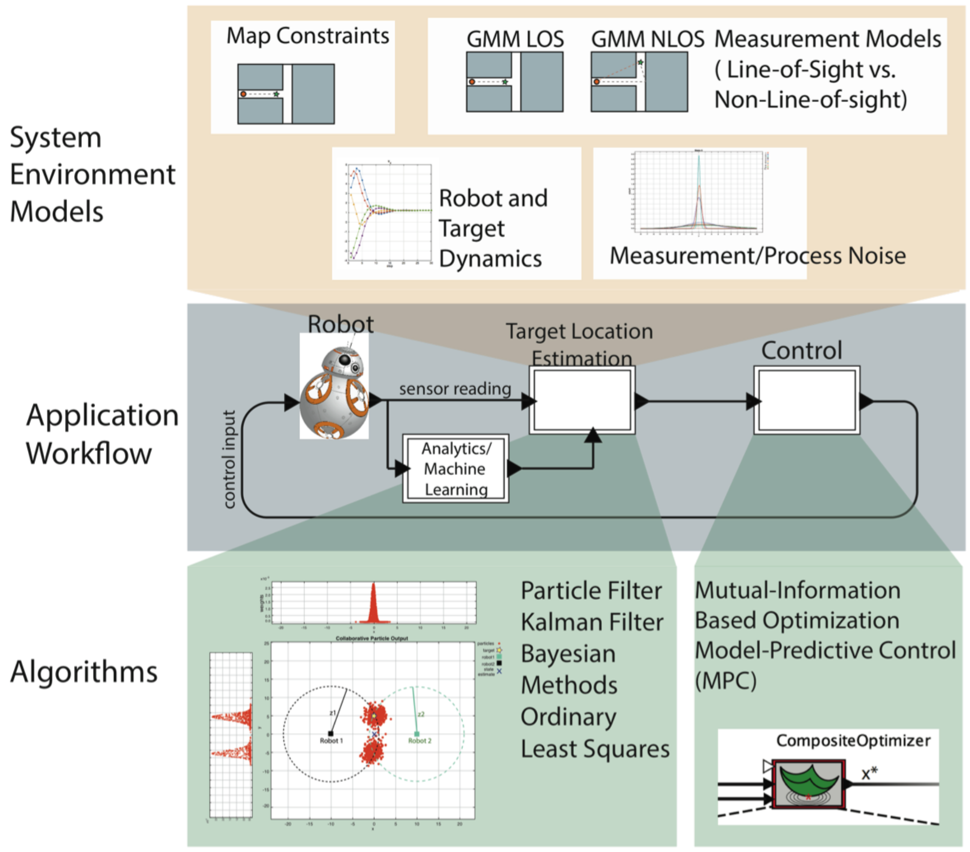

System Environment Models
Map Constraints
GMM LOS
GMM NLOS
Measurement Models (Line-of-Sight vs. Non-Line-of-sight)
Robot and Target Dynamics
Measurement/Process Noise
Application Workflow
Robot
Target Location Estimation
Control
sensor reading
control input
Analytics/ Machine Learning
Algorithms
Particle Filter
Kalman Filter
Bayesian Methods
Ordinary Least Squares
Mutual-Information Based Optimization
Model-Predictive Control (MPC)
CompositeOptimizer
x*

Figure 5.13: Aspect-Oriented Application Design in PILOT

distance. The four parameters of the modal measurement model are often obtained using supervised training techniques. Although the true LOS/NLOS noise distributions may not be unimodal and model fidelity can be improved using higher level models, several studies have reported little to no improvement when using higher level models [33, 37]. Such modal dependence is easily represented by the aspect mechanism, where multiple measurement models that are decorated by the same state-space can be present in the model. Figure 5.14 illustrates modeling of such scenario for a single range sensor.

5.2.3.2 Aspects as Decorations of DBNs

Given a modular representation of multi-modal measurements, it is often not possible to know deterministically which mode the sensor is operating in when a mobile robot system is in operation. The aforementioned example of inferring line-of-sight conditions is one example where different measurement statistics need to be leveraged for a statistical inference of the mode of operation. Although geometric

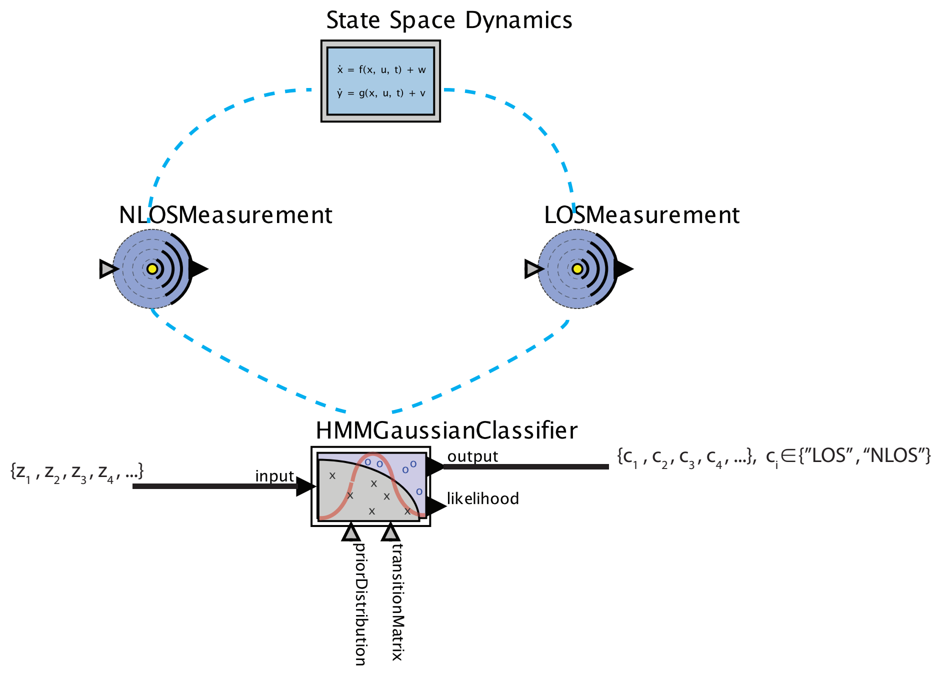


Figure 5.14: Multi-Modal Aspect Association

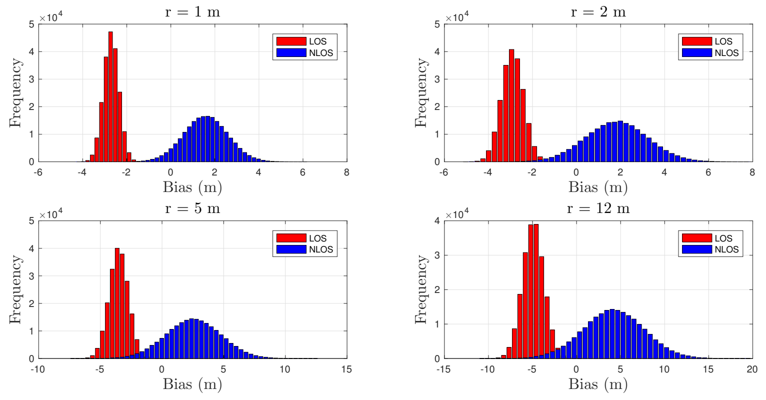


Figure 5.15: Simulated Histograms for Range Measurement Bias Under LOS and NLOS Conditions

LOS could be considered as a deterministic solution, doing so becomes tremendously inefficient in real-time. Furthermore, since geometric LOS is different from RF LOS, using this method requires higher-fidelity computations of actual RF LOS, which becomes intractable in most scenarios [32].

As suggested by the structure of the LOS and NLOS noise distributions given by (5.14) and Figure 5.15, the use of Hidden Markov Models (HMMs) for inference of the line-of-sight condition becomes a viable option. The purpose of the

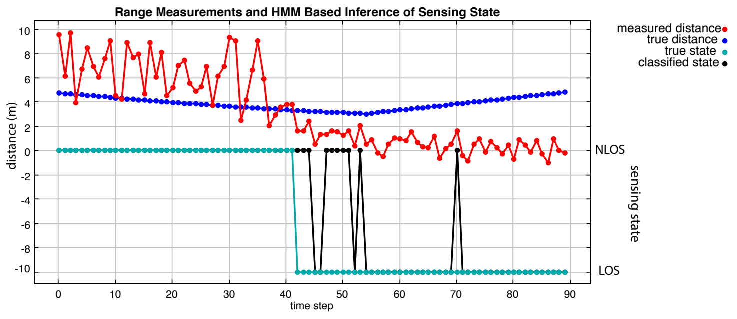


Figure 5.16: HMM Classification Results for an NLOS-to-LOS Experiment

HMM is to yield a sequence of classification labels $c_i, c_i \in \{\text{NLOS}, \text{LOS}\}$, given raw noisy range measurements. In the example given by Figure 5.15, the Bayesian Network model is a so-called binary HMM that has two states, which are specified by two Gaussian distributions, provided by the `Measurement Model` aspects called `NLOSMeasurement` and `LOSMeasurement`, respectively. Note that these actors implement the `TrainableGaussianModel` interface as described in the previous section. The DBN actors are implemented to return the *name* of the decorator as the classification label, which can be processed by downstream actors to decide on the valid aspect association at a given simulation step.

An example trace is given by Figure 5.16, which illustrates the true vs. classified sensing state, as well as the raw range measurement provided to the HMM. In the experiment, sensing state changes from no-line-of-sight (NLOS) to line-of-sight (LOS) at simulation step $t = 41$, as observed by the change of statistics of the measurements. The HMM classification is able to correctly track the true state after a transient period.

5.2.3.3 Aspects for Modeling Environmental Constraints

A widely used dynamic environmental constraint in robotics applications are Map constraints. Maps of the environment in multi-robot systems are often built via sensors on board mobile robots, and are constantly subject to change. The changes may be due to mobile obstacles in the environments, noisy sensor readings, tracking errors in robot's motion, and sensors' operating conditions. We consider representing the sensed map of the environment as a `Map` aspect. Any relevant downstream actor decorated by a `Map` aspect can therefore be subject to dynamic Map constraints in its operation. As an example, a State Space Simulator in PILOT that is decorated by a Map aspect will constrain the simulated state to remain within the current map area. Similarly, a Particle Filter decorated by a `Map` aspect will limit particle generation in a valid area and avoid spurious state estimates that are outside the map.

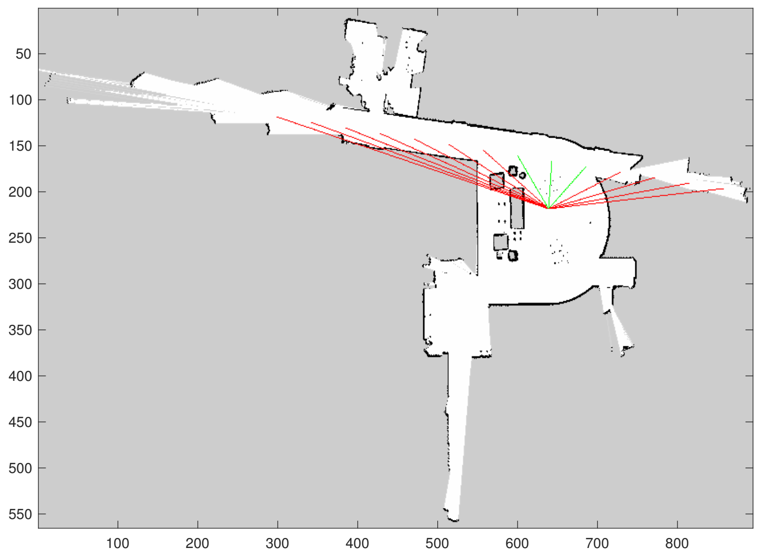


Figure 5.17: Example geometric LOS on a sensed map of the DOP Center [green: line-of-sight, red: no line-of-sight]

5.3 Aspect-Oriented Inference on SSMs

The aspect representation of dynamics and measurement models ensure modularity in state-space representation, as well as provides operational meaning to learning and classification tasks on DBNs. Next, we will discuss how these aspects are used in conjunction with executable quantities, especially with state inference and optimization actors. Table 5.2 summarizes the current aspect library with the respective aspect associations that they support. We will next focus on a subset of these tasks in detail and explain their use in robotics applications.

5.3.1 Particle Filtering

Even though dynamics and sensing can be modeled quite accurately using deterministic, and even linear or piecewise linear models, many other problems in robotics are often too complex to fully represent via deterministic models. As an example, planning and control problems that assume complete deterministic models of robots together with their environment quickly become infeasible, as the applications involve more agents that operate at higher sampling rate, with strict timing and safety requirements [110]. A widely adopted alternative to this approach is probabilistic models, which have become popular for perception and decision making in robotics applications.

Particle filtering is a widely used tool for probabilistic inference in robotics, mainly due to its power of modeling nonparametric, multi-modal distributions of state, nonlinear dynamics, as well as successfully imposing arbitrary constraints on state and state transition during estimation. Also, representation of state with a set of particles is desirable for those applications that are subject to limited availability of computational resources, since the accuracy of the estimation can easily be traded off for real-time performance. The algorithmic derivation of a particle filter is been presented in section 4.2.4.2. In this section, we will focus on the PILOT implementation and its interaction with state-space aspects.

In cooperative robotics, estimating locations of objects, devices, and other robots is a commonly encountered problem. Moreover, the task is often carried out in environments that are sensed via imperfect sensors on board the robots, where sensor data is subject to interpretation as in the line-of-sight example discussed earlier. In such scenario with multiple sources of uncertainty, estimation of state becomes highly dependent on interpreting available data in the most accurate way possible. The particle filter actor in PILOT is designed in a way to accommodate multiple models of uncertainty at once. For this purpose, it supports association with a `StateSpaceModel`, which describes the state-space of the target to be localized via state estimation, multiple `MeasurementModel`s, that can either represent sensors on board multiple robots, or can refer to different modes of the same sensor. Moreover, environmental constraints can be associated with the filter via `Map` aspects. Note that many of these aspects can be subject to change at run time, as have been demonstrated with sensor models that can be switched according to operating conditions or trained using HMMs, and with `Map` actors, which can be updated as robots gather more information about their environment. By the help of the aspect-oriented modeling mechanism, these aspects can remain separated from the functional implementation of the particle filter, yet keep affecting the operation of the state estimation task by the help of the aspect association. While infinite streams of sensor data are processed by aspects to extract information that is only relevant to the behavior of the rest of the system, the functional model can be executed in a modular way. Figure 5.18 describes available aspect associations for a particle filter in PILOT.

5.3.1.1 Application-Specific Configurations of Particle Filtering Methods

Due to being a Sequential Monte Carlo method often representing multi-modal distributions, the accuracy of the particle filter depends highly on the number of particles, particle sampling methods, and on how well the configuration suits to the domain of application. To accommodate these sources of variability, we have built multiple configurations as part of the particle filter implementation. The application designer is able to specify the following configurations of the particle filter:

- Resampling Algorithm. The resampling step of the particle filter determines which particles will propagate to the next iteration step, as well as their im-

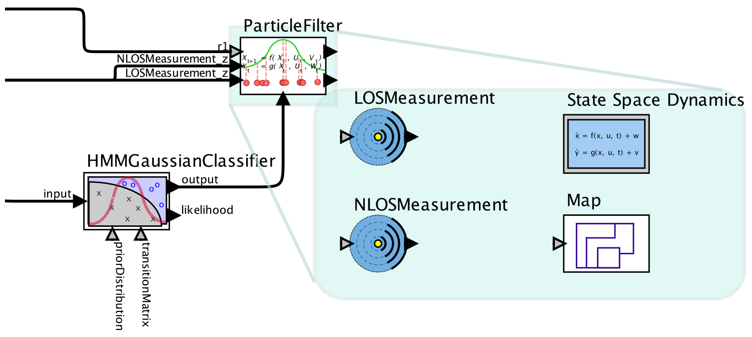


Figure 5.18: Aspect-Oriented State Estimation via Particle Filtering

portance weights. We chose to implement the two most widely used sampling methods for robot localization as part of the toolkit: *Low-variance resampling* and the *Bootstrap Particle Filter* [40].

- Estimation mode. In some applications, the particle with the highest weight can be more meaningful than the minimum-mean square error estimate. As an example, in Figure 5.20, two particle filtering schemes on range measurements is presented. Here, considering the MMSE estimate of state is not useful, as the weighted average of the particle set yields a state estimate that is not representative of the particle set, while the maximum-weight particle has a very intuitive interpretation, in being the most-likely location of the target on the map.
  - Maximum Likelihood: The state estimate is given by a weighted sum of all particles $\hat{x}_t^{MMSE} \sim \sum_{i=1}^{N} w_t^i \tilde{x}_t^i$
  - Maximum Weight: $\hat{x}_t^{MW} = \tilde{x}_t^{i*}$, where $i^* = \arg\max_{i=1\ldots N} w_t^i$
- Particle Count.
  - Internal particle count (N)
  - External particle count $(N_{out} \leq N)$

It is also key to consider the usefulness of multi-modal sensor association in the context of particle filtering. Consider the LOS/NLOS sensor models discussed earlier. Figure 5.19 plots three scenarios of state estimation via particle filtering given range-only measurements of the target. In all three cases, the particle filter is associated with a measurement model of the sensor under NLOS and LOS conditions. Given the ground-truth information about the sensing state, and using a single range-sensor, the particle filter is able to estimate the target with a MSE of 4 meters under 26 time steps. With no knowledge of true state, the MSE always remains above 9 meters.

With the use of an unsupervised classification method based on HMM inference, the filter is able to achieve a best MSE of 5.6 meters in 45 time steps.

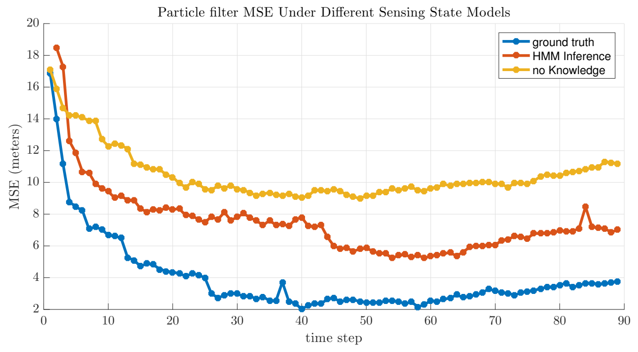


Figure 5.19: The effect of accurate sensor models on MSE of the state estimate

5.3.2 Constrained Optimization for Streaming Applications

Sensing and estimation in IoT systems is a key data-driven process. However, it is crucial for the estimated quantities to be utilized to close the feedback loop by employing meaningful control strategies. Towards achieving this goal in PILOT, we next present an actor-oriented approach to performing *optimization* on streaming data. Consider the general optimization problem

$$\begin{aligned} &\underset{\mathbf{x}\in\mathbb{R}^n}{\text{minimize}}\ f(\mathbf{x},\mathcal{Q}) \\ &\text{subject to } \mathbf{g}(\mathbf{x},\mathcal{Q}) \geq 0 \end{aligned} \tag{5.15}$$

where $f(\cdot)$ is the objective function, $g(\cdot)$ is a vector-valued constraint function, and $\mathcal{Q}$ is a vector of function parameters.

As part of PILOT, we introduce an actor interface called `CompositeOptimizer` which enables $f(\cdot)$ and $g(\cdot)$ to be defined as *actors* that operate on input tokens and upon firing, produce a scalar value for the objective function and a constraint vector evaluated at $\mathbf{x}$. The operational semantics of the actor is given by Algorithm 1. `CompositeOptimizer` is a specialized actor whose internal execution semantics is governed by the `OptimizerDirector`. This director wraps an optimization solver to be used in the optimization loop. Currently, an unconstrained optimization solver that implements a direct method algorithm named Constrained Optimization By Linear Approximation (COBYLA), and a convex solver that implements the interior-point algorithm [89]. COBYLA is a well-suited framework for generic optimization tasks, since it requires no smoothness guarantees on $f(\cdot)$ and accepts nonlinear constraint specifications. However, when the analytical expression for the cost and constraints function is available, convex solvers would perform much faster in practice. The

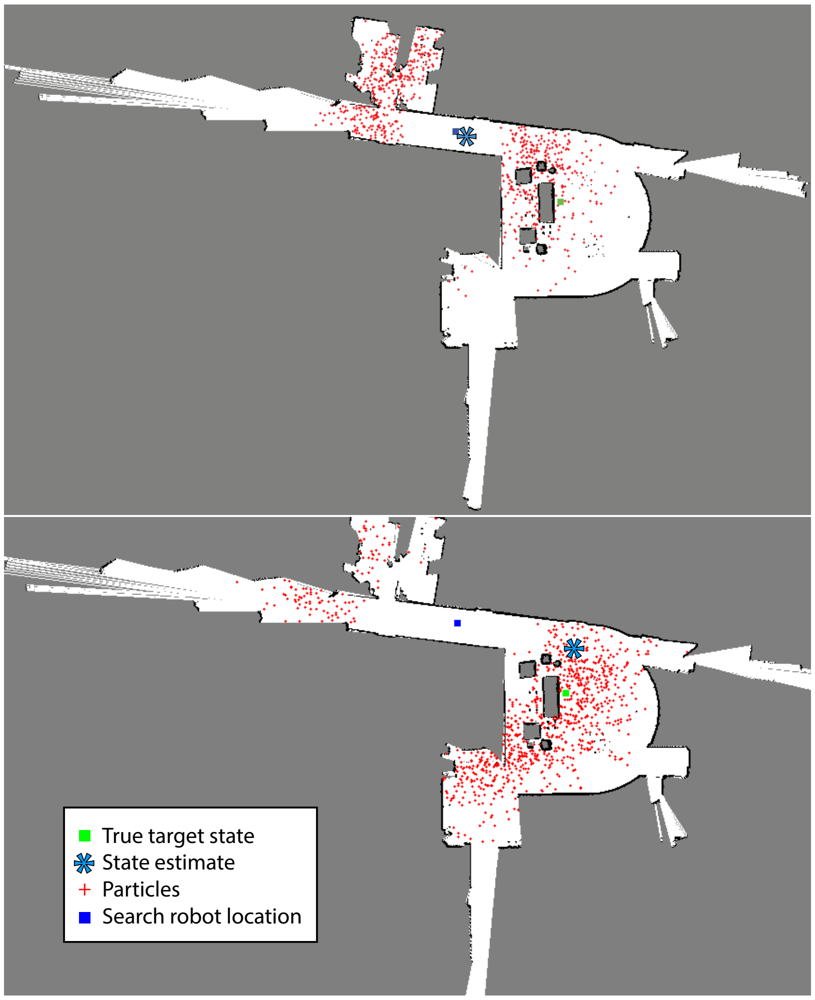


Figure 5.20: Different estimation modes for the PILOT particle filter. MMSE state estimate [top], maximum weight particle estimate [bottom]

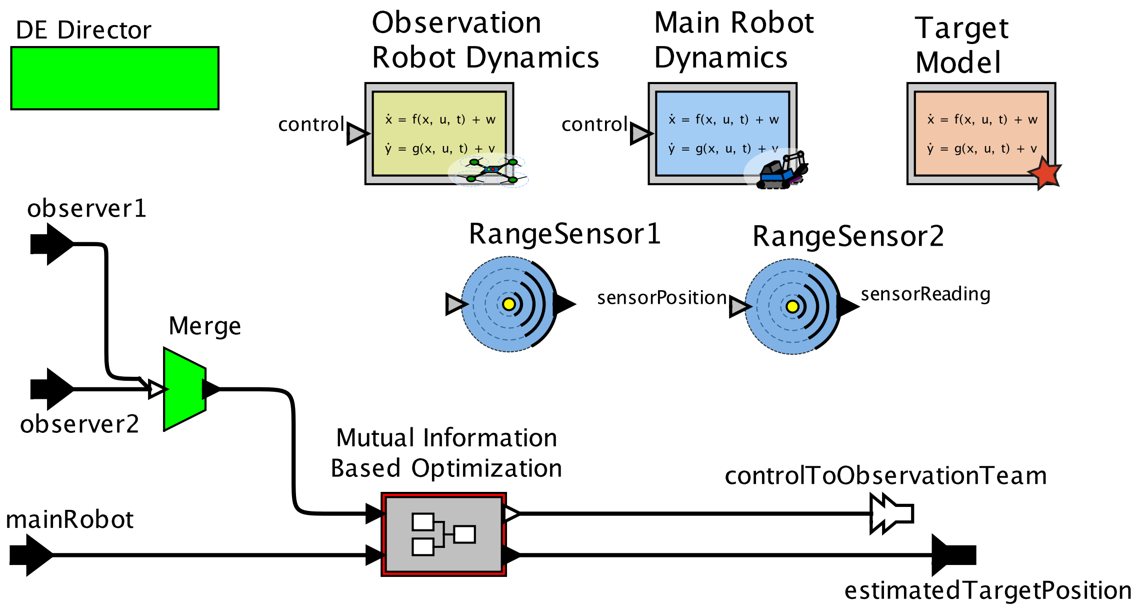


Figure 5.21: The `Observation-Optimizing Controller` demonstrates AO modeling of sensors and robot dynamics

methods `getNextX()` and `getOptimalX()` in Algorithm 1 are implemented by the designated solver.

The `CompositeOptimizer` instance given in Figure 5.22 presents an example realization of the actor that performs the stream optimization problem given by

$$\mathbf{x}^* = \arg\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}, q_1, q_2)$$
$$\text{subject to } g(\mathbf{x}, q_1, q_2) \geq 0 \tag{5.16}$$

where specific evexecutingaluations of $f(\cdot)$ and $g(\cdot)$ are computed by actors `f(x)` and `g(x)` with inputs $\{\mathbf{x}^{(k)}, q_1(i), q_2(i)\}$, where the value of optimization variable $\mathbf{x}$ at each optimization step is indexed by k, and is determined internally by the solver.

5.4 Case Study: Cooperative Mobile Robot Control

We have discussed each component of PILOT in detail. Next, we present a comprehensive application scenario that utilizes many aspects of the PILOT toolkit in modular design of cooperative robotic applications using information seeking methods for on-line decision making.

5.4.1 Problem Statement

Emerging application scenarios on robotic sensor networks include cooperative target localization and tracking, Simultaneous Localization and Mapping (SLAM), and obstacle avoidance. The use of information theoretic objectives for sensor network management has been a recent area of interest. See [119] for a complete overview of information theoretic metrics for control purposes.

In this section, we consider a cooperative multi-robot target localization scenario using robots equipped with range-only sensors. A particle filtering algorithm is used to perform target state estimation. The goal of the mobile sensor network is to successfully locate and track target position, and future goals will include pursuit objectives.

Note that in previous sections, we focused on application-specific implementations of aspect-oriented environmental constraints, multi-modal state-space models, and the combinations of these concerns for modular development of applications. In this section, we instead focus on the CPS perspective, implementing the feedback loop that obtains streaming data from mobile sensors that is used for estimation of physical quantities (i.e., robot state), which eventually is used to obtain an optimal control input to the physical system.

The target search problem can be structured with the following state-space model

$$x_0 \sim \pi_X(x_0) \tag{5.18}$$
$$z_{j_t} = \|R_{j_t} - x_t\| + \nu_t \tag{5.19}$$
$$x_{t+1} = x_t + \omega_t \tag{5.20}$$

Algorithm 1 `CompositeOptimizer` Operational Semantics

```
Input: $\mathcal{Q} \Leftarrow \mathcal{Q}_i$
Output: $\mathbf{x}^*$ that is a local optimum of $f(\cdot)$
  define P: An actor that implements SDF semantics and has inputs: $\mathbf{x}, \mathcal{Q}$ and outputs: $f, g$
  while $k < k_{MAX}$ &
        !CompositeOptimizer.converged() do
    $\mathbf{x}^{(k)} \Leftarrow$ OptimizerDirector.getNextX();
    P.readInputs($\mathbf{x} \Leftarrow \mathbf{x}^{(k)}$, $\mathcal{Q} \Leftarrow \mathcal{Q}_i$);
    P.execute();
    P.writeOutput($f(x^{(k)}, \mathcal{Q}_i) \Rightarrow f^{(k)}$,
     $g(x^{(k)}, \mathcal{Q}_i) \Rightarrow g^{(k)}$);
    OptimizerDirector.computeNextX($f^{(k)}$,$g^{(k)}$);
  end while
  $\mathbf{x}^* \Leftarrow$ CompositeOptimizer.getOptimalX();
```

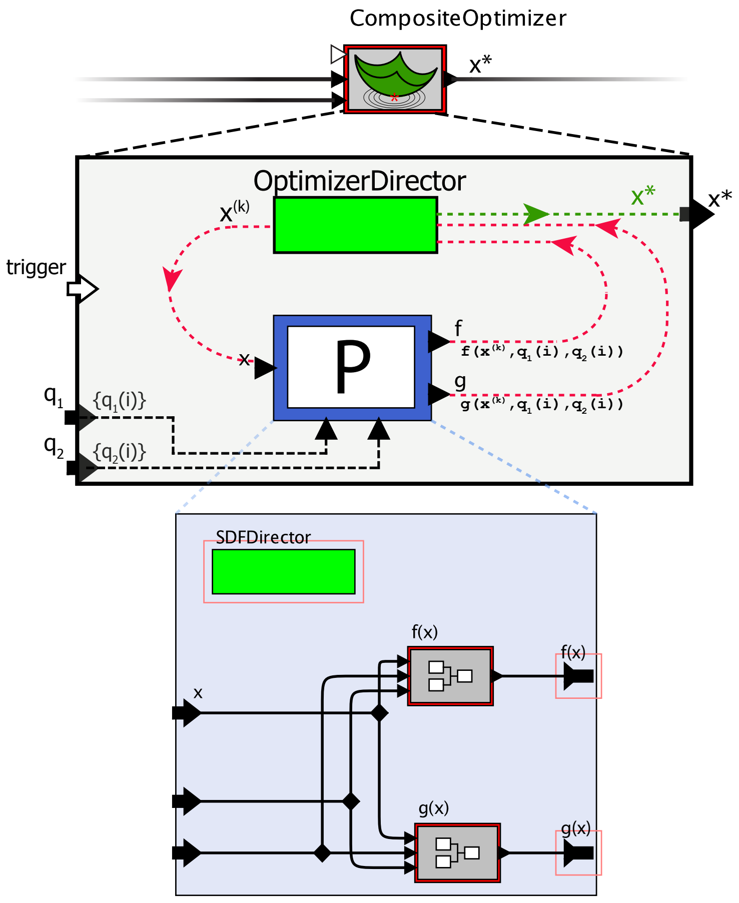


Figure 5.22: Operational Semantics of `CompositeOptimizer`

where (5.19) corresponds to a range measurement made by robot j, z_{j_t} denotes the observation made by the j'th sensor at time t, R_{j_t} is the true position of robot j at time t and ν_t is a random variable defining measurement noise.

The unknown state x is the target position in $\mathbb{R}^2$, for our case study. The state dynamics of the uncooperative mobile target is assumed to be unknown to the application, and is characterized by a random walk with process noise defined by ω_t. State estimation of the unknown target will be performed given measurements from a set of robots, and will be represented as a set of particles, $\{w_t^i, \tilde{x}_t^i\}_{i=1}^N$.

5.4.2 State-Space Aware Application Design

The traditional approach to implementing an end-to-end system for designing a target tracking application is given by (5.18)-(5.20) follows the monolithic approach of implementing a state estimator, followed by a controller, which are both implicitly dependent on the state-space of the problem.

PILOT's approach to designing such application differs from the traditional approach as it exploits the dependence of algorithmic blocks on the underlying problem state-space by defining *explicit* interfaces to the system dynamics. The target dynamics as given by (5.18) and (5.20) are contained by the `TargetDynamics` actor. The range sensors defined by (5.19)) are implemented by the `RangeSensor` actor, which is decorated by the `TargetDynamics` state-space model and therefore, can utilize the state-space parameters defined by the target and provide an imperfect measurement based on its parameters.

5.4.3 Simulation Setup

The case study assumes a network of robots equipped with range-only sensors. Simulation parameters are summarized in Table 5.3.

Simulation Parameter	Value
Size of Robot Team (M)	4
Search space	200x200 units
Sensor measurement noise	$\nu_t \sim \mathcal{N}(0, 5.0)$
Maximum Robot Speed	20 units/s
Iteration Frequency	10 Hz
Target Dynamics	Circular Motion with $\omega = \pi/5$
Target Position Prior (π_X)	Uniform over search space
Initial Target Position	$\mathbf{I}\|_{t=0} = \begin{bmatrix} -50 & 50 \end{bmatrix}$
Robot Control Inputs	$u_t^{(i)} = \begin{bmatrix} v_x & v_y \end{bmatrix},\ i \in \{0,1,2,3\}$
Robot Dynamics	$R_{it+1} = R_{it} + u_t^{(i)} \Delta t, \quad i \in \{0,1,2,3\}$

Table 5.3: Simulation Parameters

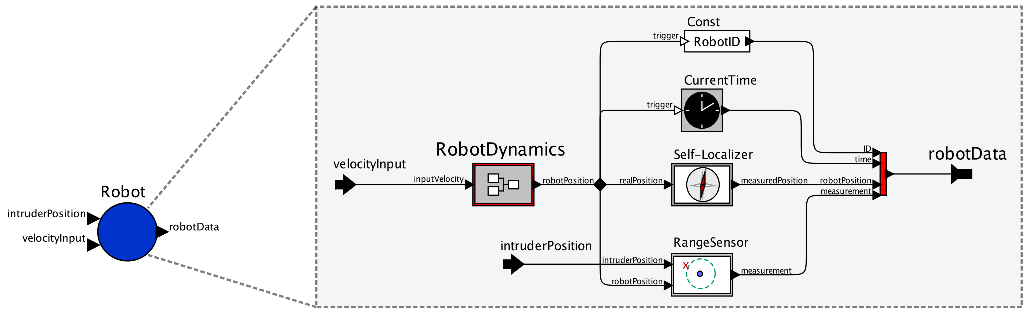


Figure 5.23: Model of the Robot Equipped with Range Sensor

The robot model is given in Figure 5.23. Note that in this model, `Robot` actors include models of range sensors with additive noise, making measurements to a target and delivering this data to a centralized computation unit. The refinement of the Target State Estimation and Control actor is depicted in Figure 5.25, which is an end-to-end PILOT model that includes state-space models of robot sensors, target dynamics, state estimation and trajectory optimization.

5.4.4 Actor-Oriented Implementation of Control Policies

Mutual information between target state and robot measurements can be used to derive a control rule to maximize the expected future mutual information, i.e., to minimize the expected future uncertainty in the target location estimate. The observers in this case are mobile robots with navigation, that expect velocity inputs from a centralized controller. The proposed objective function is given by

$$\mathbf{u}^* = \arg\max_{\mathbf{u}_\tau} I(z_\tau; x_\tau) \tag{5.21}$$

$$s.t.\ ||u_t^{(i)}|| \leq V_{max},\ i = 1, 2, ..., M \tag{5.22}$$

$$I(\mathbf{z}_\tau; x_\tau) := H(\mathbf{z}_\tau) - H(\mathbf{z}_\tau | x_\tau)\,, \tag{5.23}$$

where $I(\cdot;\cdot)$ is the mutual information (MI) defined between two random variables that are its arguments, $H(\mathbf{z}_\tau)$ is the entropy of the measurement set $\mathbf{z}_\tau$, $H(\mathbf{z}_\tau|x_\tau)$ is the conditional entropy of the measurements given the state belief, $\tau = [t+1, ..., t+T]$, where T is the time horizon of the control problem, and $\mathbf{u}$ is the array of control inputs to the mobile sensors. We will focus on the T=1 case for this case study. The exact mutual information metric given by (5.23) is not scalable for large teams of robots, an issue addressed by previous work [32, 61], which offer several approximations to the problem. A summary of potential approximations of this quantity is discussed in Appendix B. In this case study, we use a Gaussian Mixture-Model based approximation of the mutual information as a cost function.

We now illustrate how PILOT enables efficient exploration of a variety of control policies. Figure 5.25 describes a `CompositeOptimizer` interface that is configured to compute MI, given a set of particles and robot positions. The Mutual Information Approximation actor computes an approximate MI between a particle set and robot trajectories, where the state-space of the robot is established via an association with the `RobotDynamics` state-space aspect, which implements the search robot dynamics, as illustrated by Figure 5.21. A subset of common control policies for robotic path planning will be discussed next, as also summarized by Table 5.4. These policies have been implemented using PILOT via the same aspect mechanism discussed earlier.

5.4.4.1 Mutual-Information Maximization

As explained in section 5.4.4, minimizing uncertainty in a particle filter target location estimate with as few iteration steps as possible (or often analogously, in the smallest possible time frame) can be achieved by a control policy that uses information based utility functions. Figure 5.26a demonstrates that when the control objective, as defined in (5.24), is to maximize MI between the set of range measurements and the particle filter target estimate, the robots tend to follow a trajectory that spreads them apart, such that the *uncertainty* in the collective target estimation problem is *minimized*.

$$\begin{aligned} \mathbf{u_t}^* &= \arg\max_{\mathbf{u}_t \in \mathcal{U}^M} \; I(\mathbf{z}_{t+1}; x_{t+1}) \\ \text{s.t } &||u_t^{(i)}|| \leq V_{max}, \; i = 1, 2, ..., M \end{aligned} \tag{5.24}$$

5.4.4.2 MI Maximization with Single Pursuer

Upon successful tracking of the target, pursuit will be a common follow-up scenario in mobile robot swarm applications. Equation (5.25) presents a control policy for a robot swarm that steers the robot team to maximize MI, equivalently, to minimize uncertainty in the target position estimate, while assigning the robot closest to the target estimate to be the pursuer.

$$\begin{aligned} \mathbf{u}_t{}^{(i)*} &= \begin{cases} \arg\min\limits_{\mathbf{u}_t{}^{(i)}} \|R_{t+1} - x_{t+1}\| & \text{if } \|R_{it} - x_t\| \leq \|R_{j_t} - x_t\| \; \forall j \neq i \\ \arg\max\limits_{\mathbf{u}_t{}^{(i)}} \; I(z_{t+1}^{(i)}, x_{t+1}) & \text{otherwise} \end{cases} \\ \text{s.t } &||u_t^{(i)}|| \leq V_{max}, \; i = 1, 2, ..., M \end{aligned} \tag{5.25}$$

5.4.4.3 Direct Target Pursuit

The final pursuit scenario considers a direct-pursuit policy, where the controller optimizes robot control signals to minimize the Euclidean norm between the robot positions and the target state estimate in the next time step.

$$\begin{aligned} \mathbf{u_t}^* &= \arg\min_{\mathbf{u}_t} \; ||\mathbf{R}_{t+1} - x_{t+1}|| \\ \text{s.t } &||u_t^{(i)}|| \leq V_{max}, \; i = 1, 2, ..., M \end{aligned} \tag{5.26}$$

Execution traces from each of these control policies are given in Figure 5.26, which assume the parameter space given by Table 5.3. Note that the path planning problem defined by the PILOT model is solely parameterized the choice of the objective function and constraints, and requires no modification to the state-space model itself.

5.4.5 Imperfect Communication of Sensor Data

In addition to facilitating exploration of control policies, PILOT enables exploration of the effects of implementation realities. We illustrate this by considering

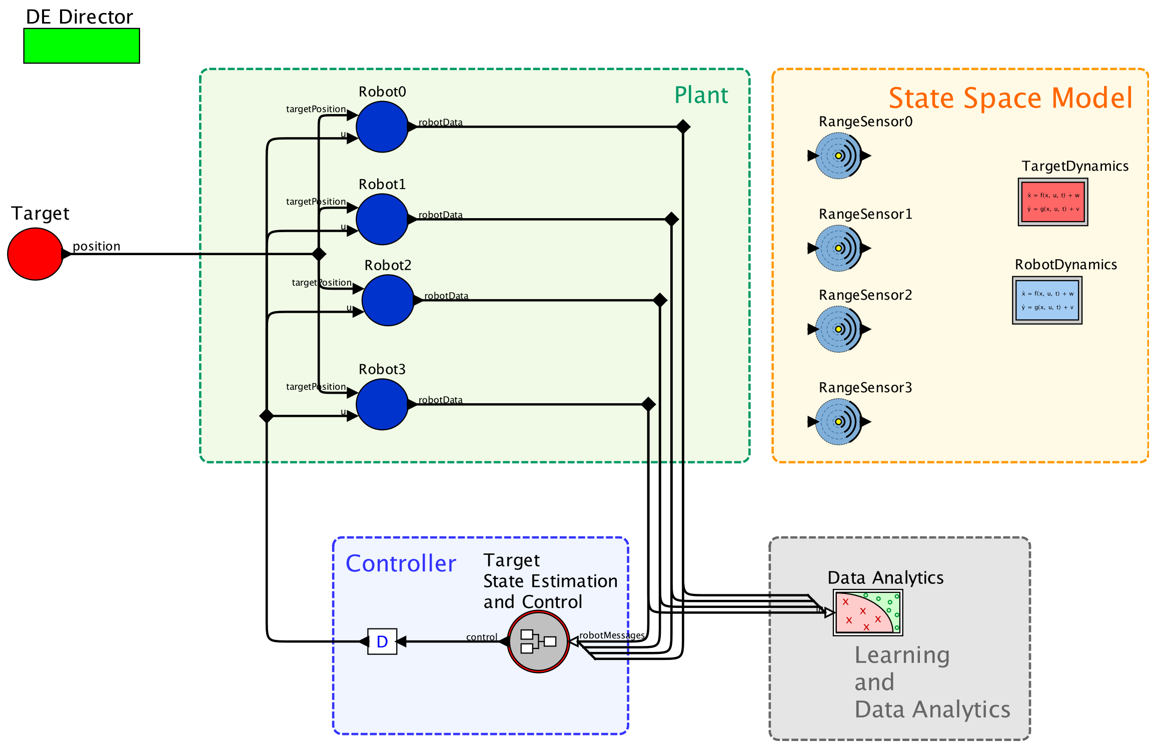


Figure 5.24: Top-Level Model for Range-Only Target Localization

Control Policy	Optimization Problem
MI Maximization	$\mathbf{u_t}^* = \arg\max_{\mathbf{u}_t \in \mathcal{U}^M} I(\mathbf{z}_{t+1}; x_{t+1})$ s.t (B.2) holds
MI Maximization with Single Pursuer	$\mathbf{u}_t{}^{(i)*} = \begin{cases} \arg\min_{\mathbf{u}_t{}^{(i)} \in \mathcal{U}} \lVert \mathbf{R}_{t+1} - x_{t+1} \rVert \\ \text{s.t (B.2) holds} & \text{if } d_t{}^{(i)} < d_t{}^{(j)}, \\ & \forall j \neq i \\ \arg\max_{\mathbf{u}_t{}^{(i)} \in \mathcal{U}} I(z_{t+1}^{(i)}; x_{t+1}) \\ \text{s.t (B.2) holds} & \text{otherwise} \end{cases}$ $d_t{}^{(i)} := \lVert R_{i\,t} - x_t \rVert,\ i \in \{1, 2, ..., M\}$
Direct Target Pursuit	$\mathbf{u_t}^* = \arg\min_{\mathbf{u}_t \in \mathcal{U}^M} \lvert\lvert \mathbf{R}_{t+1} - x_{t+1} \rvert\rvert$ s.t (B.2) holds

Table 5.4: Summary of Control Policies for Path Planning

network packet loss. Consider a communication channel that exhibits a probabilistic packet drop behavior, where the packet drop probability is defined by a Bernoulli

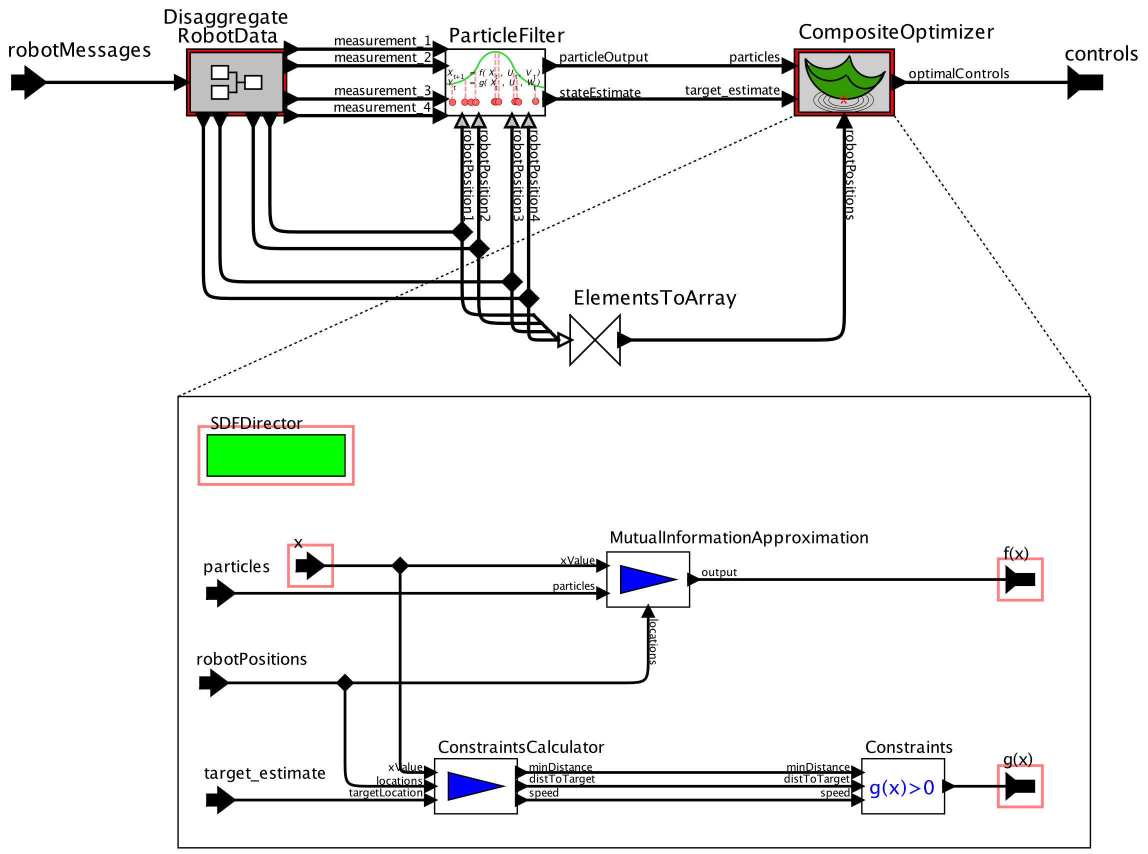


Figure 5.25: PILOT Model for Target State Estimation and Trajectory Optimization

random variable with parameter p. Such a communication channel is often referred to as a packet erasure channel. For each packet transmitted through the channel, $p(\texttt{packetDropped}) = p$.

In the PILOT application described in Figure 5.24, we configure the `NetworkModel` actor to exhibit this probabilistic channel loss behavior, where packets from each robot are transmitted through an individual dedicated link. Under the mutual-information maximization control policy presented in section 5.4.4.1, which is considered to yield the least average MSE in target position, the effect of probabilistic loss of measurements is analyzed. The results are given in Figure 5.27. Notice that even with a 50% loss from each robot, target state estimation can be performed effectively, nearly as accurately as the no loss case.

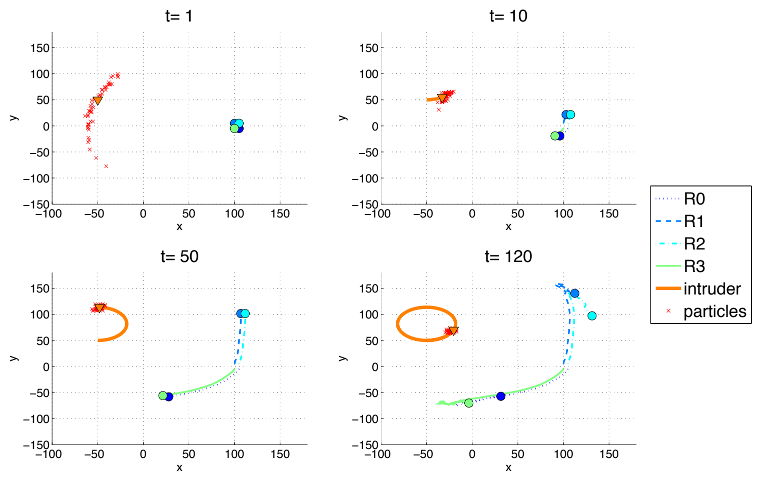
Figure 5.26a will help explain why a team of 4 robots are able to tolerate a 50% loss and still maintain a state estimate with error close to the single-sensor standard deviation level. Notice that robots, under the mutual-information maximization control policy, are moving in pairs, with very similar trajectories. A probabilistic loss of half the messages at each time step still enables the particle filter to receive one range measurement from each robot pair that is traveling relatively close together. The channel loss example demonstrates that spatial redundancy of sensors can compensate for a highly unreliable communication channel.

We finally investigate the robustness of the considered control policies in the presence of imperfect communication channels. Figure 5.28 illustrates the effect of probabilistic channel loss on the average steady-state MSE of three control policies. It is observed that trajectories obtained for MI maximization and MI-maximization with single pursuer remain marginally affected for

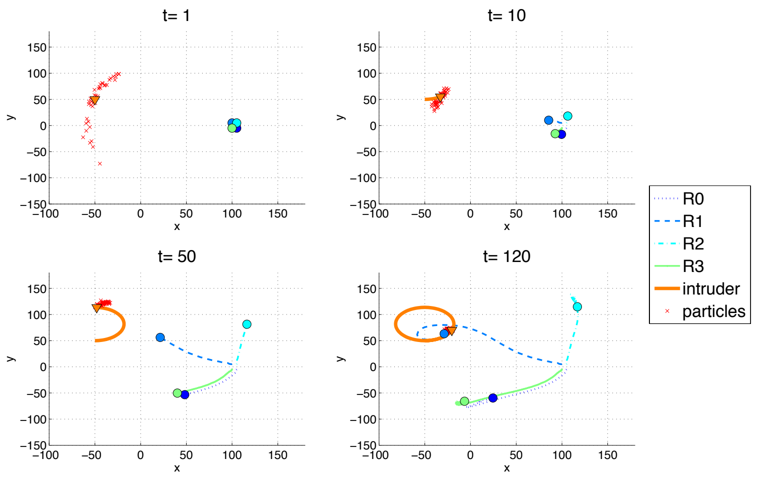
5.5 Conclusion

In this chapter, we present PILOT, a novel toolkit for modular, data-driven, actor-oriented design of machine learning and optimization algorithms for robotic sensor network applications. We develop an actor interface to a collection of algorithms, explain how separation of concerns and modular design can be ensured by the use of aspect-oriented modeling. In particular, we introduce parameter estimation, classification, sensor and dynamics modeling capabilities in PILOT, and also describe implementation of estimation and optimization algorithms using the toolkit.

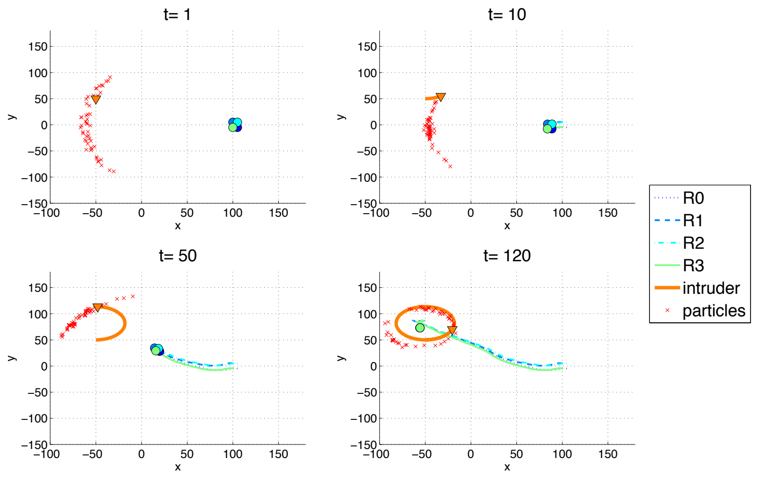
We illustrate use of the toolkit with a robot swarm target localization example that requires on-line target state estimation with a variable set of sensor inputs. Additional case studies that extend the work presented in this chapter are available, which in particular analyze the aspects of resiliency of the robot swarms in the presence of uncertainty [42, 43] and introduce Unscented Kalman Filter based extensions to the proposed techniques.



(a) Sample robot trajectory for the MI Maximization Control Policy



(b) Sample robot trajectory for MI Maximization with Single Pursuer



(c) Sample robot trajectory for Direct Target Pursuit

Figure 5.26: Simulation Traces for Different Trajectory Control Policies

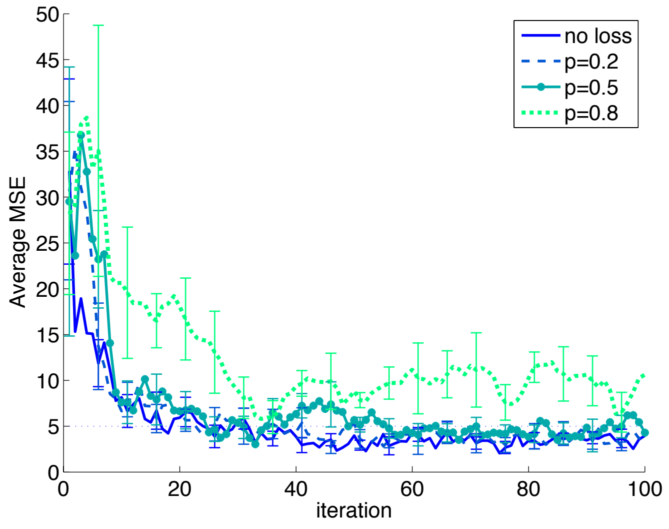


Figure 5.27: Cooperative localization error of the Mutual-Information Maximization Policy in the presence of probabilistic channel loss

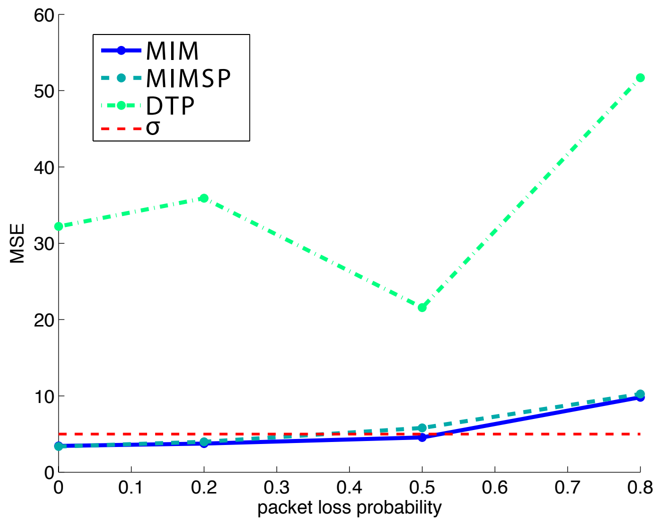


Figure 5.28: The effect of probabilistic channel loss on algorithmic accuracy for target state estimation [MIM: Mutual-Information Maximization, MIMSP: MI Maximization with Single Pursuer, DTP: Direct Target Pursuit]

Chapter 6

Data-Driven Controller Design via Control Improvisation

We have previously discussed that significant barriers exist towards dependable and reliable data-driven techniques. In particular, data-driven methods are difficult to verify and validate in design time, rendering them unfit for safety-critical applications. In this chapter, we will present a recent formalism and its application to data-driven home automation, where we show how data-driven controllers can be combined with formal methods for reliable data-driven control.

We consider the problem of generating randomized control sequences for complex networked systems typically actuated by human agents. Our approach leverages a concept known as *control improvisation*, which is based on a combination of data-driven learning and controller synthesis from formal specifications [50, 51]. We learn a generative model from streaming sensor data (for instance, an explicit-duration hidden Markov model, or EDHMM), and then restrict the learned model in order to guarantee that the generated sequences satisfy some desirable specifications given in Probabilistic Computation Tree Logic (PCTL). We present an implementation of our approach and apply it to the problem of mimicking the use of lighting appliances in a residential unit, with potential applications to home security and resource management. We present experimental results showing that our approach produces realistic control sequences, similar to recorded data based on human actuation, while satisfying suitable formal requirements.

6.1 Introduction

The promise of the emerging Internet of Things (IoT) is to leverage the programmability of connected devices to enable applications such as connected smart vehicles, occupancy-based automated HVAC control, autonomous robotic surveillance, and much more. However, this promise cannot be realized without better tools for the automated programming and control of a network of devices — computational platforms, sensors, and actuators. Traditionally, this problem has been approached from two different angles. The first approach is to be *data-driven*, leveraging the ability of devices and sensors to collect vast amounts of data about their operation and environments, and using learning algorithms to adjust the working of these devices to optimize desired objectives. This approach, exemplified by devices such as smart learning thermostats, can be very effective in many settings, but typically cannot give any guarantees of correct operation. The second approach is to be *model-driven*, where accurate models of the devices and their operating environment are used to define a suitable control problem. A controller is then synthesized to guarantee correct operation under specified environment conditions. However, such an approach is difficult in settings where such accurate models are hard to come by. Moreover, strong correctness guarantees may not be needed in all cases.

Consider, for instance, the application domain of home automation. More specifically, consider a scenario where one is designing the controller for a home security system that controls the lighting (and possibly other appliances) in a home when the occupants are away. One might want to program the system to mimic typical human behavior in terms of turning lights on and off. As human behavior is somewhat random, varying day to day, one might want the controller to exhibit random behavior. However, completely random control may be undesirable, since the system must obey certain time-of-day behavioral patterns, and respect correlations between devices. For these requirements, a data-driven approach where one learns a randomized controller mimicking human behavior seems like a good fit. It is important to note, though, that such an application may also have constraints for which provable guarantees are needed, such as limits on energy consumption being obeyed with high probability, or that multiple appliances never be turned on simultaneously. A model-based approach is desirable for these. Thus, the overall need is to *blend data and models* to synthesize a control strategy that obeys certain *hard constraints* (that must always be satisfied), certain *soft constraints* (that must be "mostly satisfied") and certain *randomness requirements* on system behavior.

This setting has important differences from typical control problems. For example, in traditional supervisory control, the goal is typically to synthesize a control strategy ensuring that certain mathematically-specified (formal) requirements hold on the entity being controlled (the "plant"). Moreover, the generated sequence of control inputs is typically completely determined by the state of the plant. Predictability and correctness guarantees are important concerns. In contrast, in the home automation application sketched above, predictability is not that important. Indeed, the system's behavior must be random, within constraints. Moreover, the source of randomness (behavior of human occupants) differs from home to home, and so this cannot be pre-programmed.

This form of randomized control is suitable for human-in-the-loop systems or applications where randomness is desirable for reasons of security, privacy, or diversity. Application domains other than the home automation setting described above include microgrid scheduling [107, 53] and robotic art [25]. In the former, randomness can provide some diversity of load behavior, hence making the grid more efficient in terms of peak power shaving and more resilient to correlated faults or attacks on deterministic behavior. For the latter case, there is growing interest in augmenting human performances with computational aids, such as in automatic synthesis and improvisation of music [39]. All these applications share the property of requiring some randomness while maintaining behavior within specified constraints. Additionally, the human-in-the-loop applications can benefit from data-driven methods. Streams of time-stamped data from devices can be used to learn semantic models capturing behavioral correlations amongst them for further use in programming and control.

In this chapter, we present how a recently-proposed formalism termed *control improvisation* [51] can be suitably adapted to address the problem of randomized control for IoT systems. We consider the specific setting of a system whose components can be controlled either by humans or automatically. Human control of devices generates data comprising streams of time-stamped events. From such data, we show how one can learn a nominal randomized controller respecting certain constraints present in the data including correlations between behavior of interacting components. We also show how additional constraints can be enforced on the output of the controller using temporal logic specifications and verification methods based on model checking [26, 72]. We apply our approach to the problem of randomized control of home appliances described above. We present simulated experimental results for the problem of lighting control based on data from the UK Domestic Appliance-Level Electricity (UK-DALE) dataset [68]. Our approach produces realistic control sequences, similar to recorded data based on human actuation, while also satisfying suitable formal requirements.

6.2 Background

We introduce relevant background material that the present work builds upon and establish notation for use in the rest of the chapter.

6.2.1 Discrete-Event Systems with Hidden States

Our work focuses on control of systems whose behavior can be described by a sequence of timestamped *events*. An event e is a tuple $\langle \tau, v \rangle \in T \times V$, where T is a totally ordered set of time stamps and V is a finite set of values. We define a *signal* to be a set of events, where T imposes an ordering relation on the events occurring within the signal [82].

We define the *state* of such a system to take values from a finite set of distinct

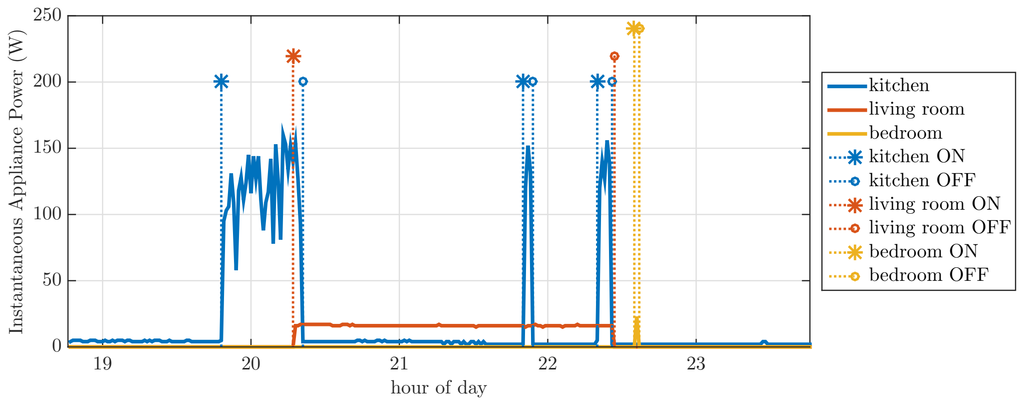


Figure 6.1: Sample appliance power trace

states, where *events* are emitted by state transitions. In many systems, the underlying events and states are hidden, and all that can be observed is some function of the state. We term this the *observation*. This function can be time-dependent and probabilistic, so that a single state can produce many different observations. We assume that the number of possible observation values is finite (this can be enforced in continuous systems by discretization), and that observations are made at discrete time steps. A sequence of observations over time generated by a behavior of the system is called a *trace*.

An example of such a trace that captures the power consumption of three appliances is given in Figure 6.1. The events related to each appliance, which can either be an "ON" or an "OFF" event in this case, are annotated on the sub-traces. Each state change of the system triggers an event. Consider, for example, that the hidden state in this scenario captures the current status of a set of physical appliances and that all appliances are initially turned off. The kitchen appliance being turned on at 19:50 pm causes an "ON" event to be emitted, and triggers a state change in the system, where in the new state, the kitchen appliance is on, and the other two appliances are off. The system stays in this state until any appliance triggers a state transition. In such a scenario, it may be that the only information available from the system are traces of the instantaneous appliance power consumptions. Given these traces, one can *infer* the state of the system and which events may have happened at particular times.

6.2.2 Control Improvisation

The *control improvisation problem*, defined formally in [51], can be seen as the problem of generating a random sequence of control actions subject to *hard* and *soft* constraints, while satisfying a *randomness requirement*. The hard constraints may, for example, encode safety requirements on the system that must always be obeyed.

The soft constraints can encode requirements that may occasionally be violated. The randomness requirement ensures that no control sequence occurs with too high probability.

This problem is a natural fit to the applications of interest in this thesis, as our end goal is to randomize the control of discrete-event systems subject to both constraints enforcing the presence of certain learned behaviors (hard constraints), and probabilistic requirements upper bounding the observations (soft constraints). In the lighting control scenario we consider later, for example, we effectively learn a hard constraint preventing multiple appliances from being toggled at exactly the same time, since this never occurs in the training data. We also use soft constraints to limit the probability that the hourly power consumption exceeds desired bounds.

More formally, the control improvisation problem is defined as follows. This is generalized from the definition in [51] to allow multiple soft constraints with different probabilities.

Definition 1 *An instance of the control improvisation (CI) problem is composed of (i) a language I of* improvisations *that are strings over a finite alphabet Σ, i.e., $I \subseteq \Sigma^*$, and (ii) finitely many subsets $\mathcal{A}_i \subseteq I$ for $i \in \{1, \ldots, n\}$. These sets can be presented, for example, as finite automata (see [51] for a thorough discussion).*

Given error probability bounds $\boldsymbol{\epsilon} = (\epsilon_1, \ldots, \epsilon_n)$ with $\epsilon_i \in [0, 1]$, and a probability bound $\rho \in (0, 1]$, a distribution $D : \Sigma^* \to [0, 1]$ with support set S is an $(\boldsymbol{\epsilon}, \rho)$- improvising distribution *if*

$$
\begin{aligned}
&(a)\ S \subseteq I && \textit{(hard constraints)},\\
&(b)\ \forall w \in S,\ D(w) \leq \rho && \textit{(randomness)},\\
&(c)\ \forall i, P[w \in \mathcal{A}_i \mid w \leftarrow D] \geq 1 - \epsilon_i && \textit{(soft constraints)},
\end{aligned}
$$

*where $w \leftarrow D$ indicates that w is drawn from the distribution D. An $(\boldsymbol{\epsilon}, \rho)$-*improviser*, or simply an* improviser*, is a probabilistic algorithm generating strings in Σ^* whose output distribution is an $(\boldsymbol{\epsilon}, \rho)$-improvising distribution. For example, this algorithm could be a Markov chain generating random strings in Σ^*. The control improvisation problem is, given the tuple $(I, \{\mathcal{A}_i\}, \boldsymbol{\epsilon}, \rho)$, to generate such an improviser.*

6.2.3 Explicit-Duration Hidden Markov Models

We have introduced the structure of an EDHMM and discussed its use cases in section 4.2.3. We will now discuss a specific construction of an EDHMM model for the domain of home automation and explain inference problems we intend to solve using this dynamic Bayesian network.

An EDHMM with discrete emissions observed for T time steps is characterized by a partially observed set of variables $(\mathbf{x}, \mathbf{d}, \mathbf{y}) = (x_1, \ldots, x_T, d_1, \ldots, d_T, y_1, \ldots, y_T)$. Each x_i indicates the hidden state of the model at time i from a finite state space

$\mathcal{X}$, which for notational convenience we assume to be the set $\{1, \ldots, N\}$. The value $d_i \in \{1, \ldots, D\}$ denotes the remaining duration in the hidden state, where D is the maximum possible state duration. Finally, each y_i is an observation drawn from a discrete alphabet $\Sigma = \{v_1, \ldots, v_M\}$. The joint probability distribution imposed by the EDHMM over these variables can be written as

$$\begin{aligned} P(\mathbf{x}, \mathbf{d}, \mathbf{y}) &= p(x_1)p(d_1) \prod_{t=2}^{T} p(x_t|x_{t-1}, d_{t-1})p(d_t|d_{t-1}, x_t) \prod_{t=1}^{T} p(y_t|x_t) \\ &= \pi_x \pi_d \prod_{t=2}^{T} p(x_t|x_{t-1}, d_{t-1})p(d_t|d_{t-1}, x_t) \prod_{t=1}^{T} p(y_t|x_t), \end{aligned}$$

where $p(x_1) \triangleq \pi_x$ and $p(d_1) \triangleq \pi_d$ are the priors on the hidden state and duration distributions, respectively. The conditional state and duration dynamics are given by

$$p(x_t|x_{t-1}, d_{t-1}) \triangleq \begin{cases} p(x_t|x_{t-1}) & \text{if } d_{t-1} = 1 \\ \delta(x_t, x_{t-1}) & \textit{otherwise} \end{cases} \tag{6.1}$$

$$p(d_t|d_{t-1}, x_t) \triangleq \begin{cases} p(d_t|x_t) & \text{if } d_{t-1} = 1 \\ \delta(d_t, d_{t-1} - 1) & \textit{otherwise} \end{cases}, \tag{6.2}$$

where $\delta(\cdot, \cdot)$ is the Kronecker delta function. Equations (6.1) and (6.2) specify the current state x_t and the remaining duration d_t for that state as a function of the previous state and its remaining duration. Unless the remaining duration at the previous state is equal to 1, the state will remain unchanged across time steps, while at each step within the state, the remaining duration is decremented by 1. When the remaining duration is 1, the next state is sampled from a transition probability distribution $p(x_t|x_{t-1})$, while the remaining duration at x_t is sampled from a state-dependent duration distribution $p(d_t|x_t)$. All self-transition probabilities are set to zero: $p(x_t|x_{t-1}) = 0$ if $x_t = x_{t-1}$. For compactness of notation, for all $x_t, x_{t-1} \in \{1, \ldots, N\}$, $d_t \in \{1, \ldots, D\}$, and $y_t \in \{v_1, \ldots, v_M\}$ we define probabilities a_{x_{t-1},x_t}, b_{x_t,y_t}, and c_{x_t,d_t} so that

$$\begin{aligned} p(x_t|x_{t-1}) &= \begin{cases} a_{x_{t-1},x_t} & \text{if } x_t \neq x_{t-1} \\ 0 & \text{otherwise} \end{cases}, \\ p(y_t|x_t) &= b_{x_t,y_t}\,, \\ p(d_t|x_t) &= c_{x_t,d_t}\,. \end{aligned}$$

We consolidate these probabilities into an $N \times N$ state transition matrix $A \triangleq (a_{ij})$, an $N \times M$ emission probability matrix $B \triangleq (b_{ij})$, and an $N \times D$ duration probability matrix $C \triangleq (c_{ij})$.

The procedure to obtain the EDHMM parameter set $\lambda = \{\pi_x, \pi_d, A, B, C\}$, given the observed sequence $\boldsymbol{y}$, is often referred to as the *parameter estimation* problem, which in the general Bayesian inference setting seeks to assign the parameters of a model so that it best explains given training data. More precisely, given a trace $(y_1, \ldots, y_T)$, parameter estimation approximates the optimal parameter set λ^* such that

$$\lambda^* = \arg\max_{\lambda} p(y_1, \ldots, y_T \mid \lambda)\,.$$

This procedure can be extended to estimate parameters from multiple traces, provided that the traces are aligned so that the first observation of each trace corresponds to the same initial state. This ensures that the state prior will be correctly captured [98]. In the case of the EDHMM, parameter estimation can be done with a variant of the well-known Expectation-Maximization (EM) algorithm for HMM. The detailed formulation is presented in [124].

6.2.3.1 EDHMM with Non-homogeneous Hidden Dynamics

The general definition of an EDHMM is useful in modeling hidden state dynamics encoded with explicit duration information. However, in many applications where the state dynamics model behaviors that exhibit seasonality, it can be useful to train separate state transition and duration distributions for different time intervals. As an example, we consider the case where the dynamics exhibit a dependence on the hour of the day, so that for each hour $h \in \{1, \ldots, 24\}$ we have different probability matrices A_h and C_h.

Estimating the parameters of an EDHMM with hourly dynamics requires an additional input sequence $\{h_1, \ldots, h_T\}$, where each $h_i \in \{1, ..., 24\}$ labels at which hour of the day the observation y_i was collected. Given the observation and hour label streams, training follows the same EM-based estimation procedure as in [124], with the difference that parameters A_h and C_h are estimated using the training data subsequences collected within hour h.

The EDHMM with hourly dynamics will be given by a parameter set $\lambda = \{\pi_x, \pi_d, \{A_h\}, B, \{C_h\}\}$, where $\{A_h\}$ and $\{C_h\}$ are the transition and duration distribution matrices valid for hour $h \in \{1, ..., 24\}$ such that

$$\begin{aligned} a^l_{i,j} &\triangleq p(x_t = j | x_{t-1} = i, d_{t-1} = 1, h_{t-1} = l) \\ c^l_{i,d} &\triangleq p(d_t = d | x_t = i, d_{t-1} = 1, h_t = l) \end{aligned}$$

where $A_l = (a^l_{ij})$ and $C_l = (c^l_{i,d})$ are the hourly transition and duration probability matrices for hour l.

6.2.4 Probabilistic Model Checking

Our approach relies on the use of a verification method known as probabilistic model checking, which determines if a probabilistic model (such as a Markov chain or Markov decision process) satisfies a formal specification expressed in a probabilistic temporal logic. We give here a high-level overview of the relevant concepts. The reader is referred to the book by Baier and Katoen [26] for further details. For our application, we employ probabilistic computation tree logic (PCTL). The syntax of this logic is as follows:

$$\begin{aligned} \phi &::= True \mid \omega \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid P_{\bowtie p}\left[\psi\right] && \text{state formulas} \\ \psi &::= \mathcal{X}\phi \mid \phi_1\, \mathcal{U}^{\leq k}\phi_2 \mid \phi_1\, \mathcal{U}\phi_2 && \text{path formulas} \end{aligned}$$

where $\omega \in \Omega$ is an atomic proposition, $\bowtie \in \{\leq, <, \geq, >\}$, $p \in [0,1]$ and $k \in \mathbb{N}$. State formulas are interpreted at states of a probabilistic model; if not specified explicitly, this is assumed to be the initial state of the model. Path formulas ψ use the *Next* ($\mathcal{X}$), *Bounded Until* $\left(\mathcal{U}^{\leq k}\right)$ and *Unbounded Until* ($\mathcal{U}$) operators. These formulas are evaluated over computations (paths) and are only allowed as parameters to the $P_{\bowtie p}\left[\psi\right]$ operator. Additional temporal operators, $\mathcal{G}$, denoting "globally", and $\mathcal{F}$ denoting "finally", are defined as follows: $\mathcal{F}\phi \triangleq True\,\mathcal{U}\,\phi$ and $\mathcal{G}\phi \triangleq \neg\,\mathcal{F}\,\neg\phi$.

We describe the semantics informally; the formal details are available in [26]. A path formula of the form $\mathcal{X}\phi$ holds on a path if state formula ϕ holds on the second state of that path. A path formula of the form $\phi_1\, \mathcal{U}^{\leq k}\phi_2$ holds on a path if the state formula ϕ_2 holds eventually at some state on that path within k steps of the first state, with ϕ_1 holding at every preceding state. The semantics of $\phi_1\, \mathcal{U}\phi_2$ is similar without the "within k steps" requirement. The semantics of state formulas is standard for all propositional formulas. The only case worth elaborating involves the probabilistic operator: $P_{\bowtie p}\left[\psi\right]$ holds at a state s if the probability q that path formula ψ holds for any execution beginning at s satisfies the relation $q \bowtie p$.

A probabilistic model checker, such as PRISM [72], can check whether a probabilistic model satisfies a specification in PCTL. Moreover, it can also compute the probability that a temporal logic formula holds in a model, as well as synthesize missing model parameters so as to satisfy a specification. We show in Sec. 6.3.5 how an EDHMM can be encoded as a Markov chain and thereby as a suitable input model to PRISM.

6.3 Control Improvisation with Probabilistic Temporal Specifications

6.3.1 Problem Definition and Solution Approach

We begin with a set of traces of a discrete-event system whose set of events is known, but whose dynamics are not. Our goal is to randomly generate new traces with similar characteristics to the given ones. Furthermore, we want to be able to enforce two kinds of constraints:

- *Hard* constraints that the traces must always satisfy, forbidding transitions between states that never occur in the input traces. For example, if no part of the input traces can be explained as a particular state transition t, then we want to assume that t is impossible and not generate any string that is only possible using it.
- *Soft* constraints that need only be satisfied with some given probability. We focus on systems whose observations are *costs*, for example power consumption, and assume soft constraints which put upper bounds on the cost at a particular time, or accumulated over a time period.

In the next section, we will formalize this problem as an instance of control improvisation. First, however, we summarize our solution approach, which consists of three main steps:

1. *Data-Driven Modeling:* From the given traces, learn an EDHMM representing the time-dependent dynamics of the underlying system. The EDHMM effectively applies hard constraints on our generation procedure by eliminating all strings assigned zero probability.
2. *Probabilistic Model Checking:* Using a probabilistic model checker, we compute the probability that a behavior of the candidate improviser obtained in the previous step will satisfy the soft constraints. If this is sufficiently high, we return the EDHMM as our generative model.
3. *Scenario-Based Model Calibration:* Otherwise, we apply heuristics that increase the probability by modifying the EDHMM parameters, and return to step (2).

A high level algorithmic workflow is given by Figure 6.2. We elaborate on each of the steps in subsequent sections.

6.3.2 Formalization as a Control Improvisation Problem

We can formalize the intuitive description above as an instance of the control improvisation (CI) problem described in Section 6.2.2. To do so, we need to specify

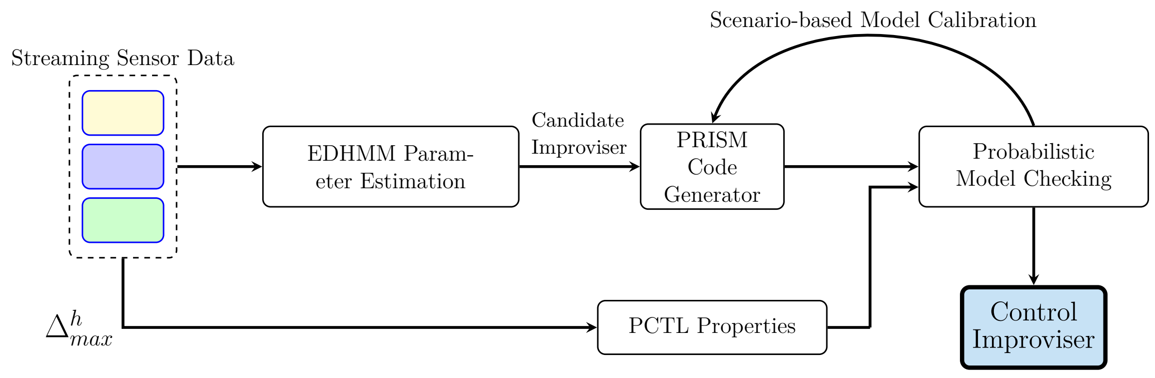


Figure 6.2: Algorithmic Workflow

the alphabet Σ, languages $\mathcal{I}$ and $\mathcal{A}_i$, and parameters ϵ_i and ρ that make up a CI instance.

Σ Since we are learning from and want to generate traces, which are sequences of observations, we let Σ be the set of all observations (i.e. those occurring anywhere in the input traces).

I We let $\mathcal{I}$ consist of all traces that are assigned nonzero probability by the EDHMM[1]. Since the CI problem requires any improviser to output only strings in I, this will ensure the hard constraints are always satisfied.

$\mathcal{A}_i, \epsilon_i$ We let $\mathcal{A}_i$ consist of all traces that satisfy the i-th soft constraint. For instance in the lighting example, $\mathcal{A}_i$ could only contain traces whose total power consumption within hour $i \in \{1, \ldots, 24\}$ of the day never exceeds a given bound. Then in the CI problem, ϵ_i is the greatest probability we are willing to tolerate of the improviser generating a trace violating the bound.

ρ We can ensure that many different traces can be generated, and that no trace is generated too frequently, by picking a small value for ρ: the CI problem requires that no improvisation be generated with probability greater than ρ, and so that at least $1/\rho$ improvisations can be generated.

This CI problem captures the informal requirements we described earlier. Now we need to show that our generation procedure is actually an improviser solving this problem according to the three conditions given in Definition 1. We consider each in turn.

[1]The definition of the CI problem given in [51] requires that I be described by a finite automaton. It is straightforward to build a nondeterministic finite automaton that accepts precisely those strings assigned nonzero probability by the EDHMM, but we will not describe the construction here since it is not needed for the technique used in this section.

6.3.2.1 Hard Constraints

By definition, any string that we generate has nonzero probability according to the EDHMM and so is in $\mathcal{I}$.

6.3.2.2 Randomness Requirement

As long as the EDHMM is ergodic (when converted to an ordinary Markov chain; see Section 6.3.5), the probability of generating any particular string $w \in \Sigma^*$ goes to zero as its length goes to infinity. So for any $\rho \in (0, 1]$, we can satisfy the randomness requirement by generating sufficiently long strings. We can efficiently detect when the EDHMM is not ergodic using standard graph algorithms, but this is unlikely to be necessary in practice for applications as lighting control.

6.3.2.3 Soft Constraints

Our procedure checks whether this requirement is satisfied using probabilistic model checking. This requires encoding the sets $\mathcal{A}_i$ as PCTL formulas, and the EDHMM as a Markov chain (explained in Sections 6.3.4 and 6.3.5 respectively). Once this has been done, the model checker computes the probability that a string generated by the EDHMM will be in $\mathcal{A}_i$. If this probability is at least $1 - \epsilon_i$, then the EDHMM satisfies the soft constraint, and if this is true for each i, it is a valid improviser. Otherwise, our procedure applies heuristics to modify the EDHMM, detailed in Section 6.3.6. As shown in that section, the heuristics decrease the expected accumulated cost, so that after sufficiently many applications the EDHMM will satisfy the soft constraints[2].

Therefore, our generation procedure yields a valid improviser solving the CI problem we defined above. We note that our technique has some further useful properties not captured by the CI problem. In particular, we can easily disable particular transitions between hidden states by setting their probabilities to zero and normalizing remaining transition probabilities appropriately. This could be useful, for example, when controlling an IoT system with unreliable components: if a component drops off the network or becomes otherwise unusable, we can disable all transitions to states in which that component is active.

6.3.3 Learning an EDHMM from Traces

The first step in our procedure is to learn an EDHMM from the input traces. Since as explained in Section 4.2.3 we use an EDHMM with different transition matrices for each hour, every input trace $\{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_T\}$ is augmented with a corre-

[2]Obviously, some soft constraints cannot be satisfied, for example one requiring that the cost at the first time step be less than the smallest possible cost of any state. See Section 6.3.6 for a precise statement.

sponding stream of labels $\{\tau_1, \ldots, \tau_T\}$ indicating the hour of the day each observation was recorded. Note that the observations need not be scalar costs, but could be vectors: for example, in our lighting experiments each observation was a K-tuple $\mathbf{y}_i = [y_{i,1}, \ldots, y_{i,K}]^T$ containing instantaneous power readings from each of K different appliances.

Given this training data, we perform EDHMM parameter estimation as described in Section 4.2.3. This yields a parameter set $\lambda = (\{A_h\}, \{C_h\}, B, \pi)$ where the matrices $\{A_h\}$ and $\{C_h\}$ give state transition and duration probabilities respectively for each hour $h \in \{1, \ldots, 24\}$. The distribution of observations for each state is given by B, and π is the prior on the state space. In this work we use categorical distributions for B and $\{C_h\}$, although in other applications it may be appropriate to use parametric distributions.

Note that the parameter estimation process based on the EM algorithm is an iterative method; thus obtaining a reasonable parameter set depends on model convergence, which in turn requires sufficient training data. In the case of an EDHMM with hourly transition matrices, if few events happen at certain hours it may not be possible to estimate some of the state transition and duration probabilities for those hours. Many application-specific heuristics exist for handling such scenarios, as outlined in [98]. The particular technique we used in our experiments is detailed in Section 6.4.1.

6.3.4 Encoding Soft Constraints as PCTL Formulas

As mentioned earlier, we consider soft constraints which put upper bounds on the cost observed at a particular time or accumulated over a time period. We illustrate how to encode upper bounds on the hourly cost — other time periods are handled analogously.

Recall that our traces take the form $\{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_T\}$ where each $\mathbf{y}_i$ is an observation, generally a vector $[y_{i,1}, \ldots, y_{i,K}]^{\mathrm{T}}$ of costs. Define $Y_i \triangleq \sum_{k=1}^{K} y_{i,k}$, the total cost at time step i. Considering that the data is sampled at the rate of N_s samples per hour, the total hourly cost accumulated up to time step t is

$$\Delta = \sum_{N_s(\lceil t/N_s \rceil - 1)+1 \leq i \leq t} Y_i.$$

In the next section, we show how a simple monitor added to the encoding of the EDHMM can maintain the value Δ.

In order to be able to impose a different upper bound Δ^h_{max} on Δ for each hour h of the day, we need to compute the current hour of the day as a function of the time step:

$$h(t) = mod(\lceil t/N_s \rceil - 1, 24) + 1\,,$$

which holds if $t = 1$ corresponds to the time step of the first sample collected within hour 1. Then we can write the soft constraint for hour h as the following PCTL formula:

$$P_{\geq 1-\epsilon_h} \, \mathcal{G} \left[(h(t) = h) \Rightarrow (\Delta \leq \Delta^h_{max}) \right]. \tag{6.3}$$

This simply asserts that with probability at least $1 - \epsilon_h$, at every time step during hour h the corresponding upper bound on Δ holds. In practice we can omit the quantifier $P_{\geq 1-\epsilon_h}$ and ask the probabilistic model checker to compute the probability that the rest of the formula holds, instead of having to specify a particular ϵ_h ahead of time.

6.3.5 Encoding the EDHMM as a Markov Chain

In this section, we discuss how the EDHMM can be represented as a Markov chain, so that the soft constraints can be verified using probabilistic model checking.

Ignoring the soft constraints for now, the interpretation of the EDHMM as a Markov chain follows the outline in Section 4.2.3: we expand the state space with a new state variable $d \in \{1, \dots, D\}$ which keeps track of the remaining duration in the current hidden state $x \in \{1, \dots, N\}$. When $d > 1$, we stay in x for another time step, decrementing d. Only when $d = 1$ do we transition to a new hidden state, picking the new value of d from the corresponding duration distribution.

Since we use an extension of the EDHMM where state transition and duration probabilities depend on the current hour, we need to expand the state space further to keep track of time. The state variable $t \in \{0, \dots, T\}$ indicates the current time step, with $t = 0$ being an initialization step in which the state is sampled from a state prior π. Note that the domain of t need not grow unboundedly with T: in our example where we use different transition probabilities for each of the 24 hours, we only need to track the time within a single day.

Finally, in order to detect when the soft constraints are violated, we need to monitor the total hourly cost Δ defined in the previous section. We add the state variable $\Delta \in \{0, \dots, \Delta_{max} + 1\}$, where Δ_{max} is the largest of the hourly upper bounds Δ^h_{max} imposed by the soft constraints. This range of values for Δ is clearly sufficient to detect when the total cost exceeds any of these bounds. Maintaining the correct value of Δ is simple: at each time step we increase it by a cost sampled from the appropriate emission distribution, except when a new hour is starting, in which case we first reset it to zero.

Putting this all together, we obtain a Markov chain whose states are 4-tuples (x, d, t, Δ) with the state variables as described above. The initial state is $(0, 1, 0, 0)$. Given the current state, the next state (x', d', t', Δ') is determined as follows:

`EDHMM:`

$$\begin{array}{rr}
(t = 0) \rightarrow & x' \sim \pi_x \;\wedge d' \sim C_{h(t)}(x') \;\wedge t' = t + 1 \\
(t > 0) \wedge (d > 1) \rightarrow & x' = x \;\wedge d' = d - 1 \;\wedge t' = t + 1 \\
(t > 0) \wedge (d = 1) \rightarrow & x' \sim A_{h(t)}(x) \;\wedge d' \sim C_{h(t)}(x') \;\wedge t' = t + 1
\end{array}$$

`Cost Monitor:`

$$\begin{aligned}
(t = 0) \rightarrow &\qquad \Delta' = 0 \\
(t > 0) \wedge (h(t') = h(t)) \rightarrow &\qquad \Delta' = \Delta + \sum_{i=1}^{K} p_i,\ \mathbf{p} \sim B(x) \\
(t > 0) \wedge (h(t') \neq h(t)) \rightarrow &\qquad \Delta' = \sum_{i=1}^{K} p_i,\ \mathbf{p} \sim B(x)\,,
\end{aligned}$$

where $h(t) = mod(\lceil t/N_s \rceil - 1, 24) + 1$.

6.3.6 Scenario-Based Model Calibration

The procedure described so far provides a way to obtain a generative model that captures the probabilistic nature of events and their duration characteristics in a physical system, and to verify that the model satisfies desired soft constraints. However, the model may not satisfy these constraints with sufficiently high probability, particularly if the constraints are not always satisfied by the training data. In terms of control improvisation, the error probability of our improviser for some soft constraint i is greater than the desired ϵ_i. We now describe two general heuristics for calibrating the EDHMM to decrease the error probability while preserving the faithfulness of the improviser to the original data. In particular, these heuristics do not introduce new behaviors: any trace that can be generated by the calibrated improviser could already be generated before calibration. Since the soft constraints we consider place upper bounds on the observed costs, both heuristics seek to decrease the costs of some behavior of the improviser.

6.3.6.1 Duration Calibration

The duration distributions of the trained EDHMM, $\{C_h\}$, assume a maximum state duration D that is enforced during the training process. One simple way to decrease cost is to further restrict the duration distributions by truncating them beyond some threshold for some or all states. An effective strategy in practice is to eliminate outliers in the duration distributions of states with high expected cost.

This heuristic has the advantage of leaving the transition probabilities of the model completely unchanged, and so is a relatively minor modification. On the other hand, it cannot reduce the duration of a state below 1 time step. So although it can eliminate some high-cost behaviors from the model, it is not guaranteed to eventually yield an improviser satisfying the soft constraints.

6.3.6.2 Transition Calibration

A different approach is to modify the state transition probabilities, making the model less likely to transition to a high cost state during certain hours of the day.

Specifically, we can limit the probability of transitioning from any state i to a particular state x_r during hour h_r to be at most some value p_r^i. We shift the removed probability mass to the transition leading to the state x_{min} with least expected cost, which we assume is strictly less than that of x_r. Writing the original transition probability matrix A_{h_r} as (a_{ij}), we replace it in the EDHMM with a new matrix $\tilde{A}_{h_r} = (\tilde{a}_{ij})$ defined by

$$\tilde{a}_{ij} = \begin{cases} \min(p_r^i, a_{ij}) & \text{if } j = x_r \\ a_{ij} + (a_{ix_r} - \min(p_r^i, a_{ij})) & \text{if } j = x_{min} \\ a_{ij} & \text{otherwise.} \end{cases}$$

Note that the second case ensures that the transition probabilities from any state $i \in \mathcal{X}$ are properly normalized. Provided that the limits p_r^i are chosen such that $\tilde{a}_{ix_r} < a_{ix_r}$ for some $i \in \mathcal{X}$, the heuristic will decrease the expected cost of a behavior generated by the improviser.

Applying the heuristic iteratively for every choice of $x_r \neq x_{min}$ and hour $h_r \in \{1, \ldots, 24\}$ will eventually result in an improviser that remains at the x_{min} state for all time steps (assuming x_{min} is the starting state). Thus for any soft constraints which are true for behaviors that only stay at x_{min}, our procedure will eventually terminate and yield a valid improviser. This over-simplified improviser is unlikely to model the original data well, but it is only attained as the limit of this heuristic: in practice, judicious choices of the state x_r and limits p_r^i can improve the error probability significantly in a few iterations without drastically changing the model.

6.4 Experimental Results and Analysis

6.4.1 Experimental Setup

To demonstrate the control improvisation approach we have described in Sec. 6.3, we use the UK Domestic Appliance-Level Electricity (UK-DALE) dataset [68], which contains disaggregated time series data representing instantaneous power consumptions of residential appliances from 5 homes over a period of 3 years.

We consider a lighting improvisation scenario over the three most-used lighting appliances in a single residence, each from a separate room of the house. The data is presented as a vector-valued power consumption sequence $\mathbf{y}$ with a corresponding sequence of time stamps $\boldsymbol{\tau}$. The input stream $\mathbf{y} = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_T\}$ consists of 3-tuples

$$\mathbf{y}_i = \begin{bmatrix} y_{i,1} \\ y_{i,2} \\ y_{i,3} \end{bmatrix}, \; i = 1, \ldots, T,$$

where the values $y_{i,1}$, $y_{i,2}$, and $y_{i,3}$ are instantaneous power readings with time stamp τ_i from the main kitchen light, a dimmable living room light, and the bedroom light respectively. The power readings were sampled with a period of 1 minute and are measured in watts.

In our experiments, we synthesized three improvisers from this data: one using an unmodified EDHMM, and two that were calibrated using the different kinds of heuristics described in Section 6.3.6 to enforce soft constraints on hourly power consumption. Below, we describe the specific choices that were made when implementing each of the three main steps of our procedure.

6.4.1.1 Data-Driven Modeling

We assume there are three sources of hidden events, corresponding to each of the three appliances being turned on or off. This yields a hidden state space $\mathcal{X}$ with 8 states, one for each combination of active appliances. Based on inspection of the dataset, we chose the maximum state duration to be 720 time steps (12 hours, sufficient to allow long periods when all appliances are off). Since we used disaggregated data, our observations are 3-tuples of power consumptions (quantized to integer values as part of the dataset), which we assume fall in the alphabet $\Sigma = \Sigma_1 \times \Sigma_2 \times \Sigma_3$ where $\Sigma_1 = \{0, 1, \ldots, 350\}$, $\Sigma_2 = \{0, 1, \ldots, 20\}$, and $\Sigma_3 = \{0, 1, \ldots, 30\}$ (the maximum consumptions for each appliance were again obtained by inspecting the dataset). Having fixed these parameters (summarized in Table 6.1), an EDHMM was trained from a 100-day subset of the data from one residence. Several portions of this training data (for one appliance) are shown at the top of Figure 6.8.

Note that for the specific case of lighting improvisation, since the power emission distributions of each appliance are independent, $B \triangleq p(\mathbf{y_t}|x_t)$, the learned emission probability matrix over vector-valued observations, can be written as

$$B = p(\mathbf{y_t}|x_t) = \prod_{k=1}^{K} p(y_{t,k}|x_t)\,.$$

It should also be noted that following the training process, some of the state transition probabilities $\{A_h\}$ may remain unlearned, i.e., we may have

$$\sum_{j=1}^{N} a_{i,j}^{h} = 0$$

for some state $i \in \{1, ..., N\}$. This can occur, for example, when no state transitions from state i happen during the hour h in any of the input traces. Since it is key to capture the observed appliance behavior, we treat these incomplete distributions that are unobserved in the training data as behaviors that should also be absent from the set of improvised behaviors. Consequently, we use a completion strategy that

Parameter ID	**Value**
Data Source	UK DALE Dataset
House ID	house_1
Appliance IDs	kitchen_lights livingroom_s_lamp bedroom_ds_lamp
Training Duration	100 days
Training Start Date	30 Jul 2013 19:07:56 GMT
Sampling Period (T_s)	60 s
Training Sequence Length (T)	144000
Maximum State Duration (D)	720
Appliance 1 Costs (Σ_1)	$\{0, 1, \ldots, 350\}$
Appliance 2 Costs (Σ_2)	$\{0, 1, \ldots, 20\}$
Appliance 3 Costs (Σ_3)	$\{0, 1, \ldots, 30\}$
State Labels	OFF: All appliances off K: Kitchen on L: Living room on B: Bedroom on KL: Kitchen and living room on KB: Kitchen and bedroom on LB: Living room and bedroom on KLB: All appliances on

Table 6.1: Parameters of the training dataset for EDHMM learning

forces transitions to the state x_{min} with the least expected cost (i.e. the state with all appliances off) in this scenario:

$$\begin{gathered}\forall a_{i,j}^h\,,\ \text{where} \sum_{j=1}^{N} a_{i,j}^h = 0,\\ i,j \in \{1,...,N\},\ h \in \{1,...,24\},\\ \tilde{a}_{i,j}^h = \begin{cases} 1 & \text{if } j = x_{min} \\ 0 & \text{otherwise} \end{cases}\end{gathered}$$

where $\tilde{a}_{i,j}^h$ is the adjusted state transition probability of switching from state i to j in hour h. Note that in this case study, such incomplete parameter estimates arose only for early morning hours in which few state transitions were recorded (typically hours $h \in \{1, \ldots, 5\}$). Having completed the transition probability matrices in this way, we obtain a fully specified EDHMM.

6.4.1.2 Probabilistic Model Checking

We experimented with soft constraints upper bounding the total power consumed during each hour. Figure 6.3 depicts the hourly energy consumptions of each appliance, as well as the aggregated consumption, averaged across each day in the

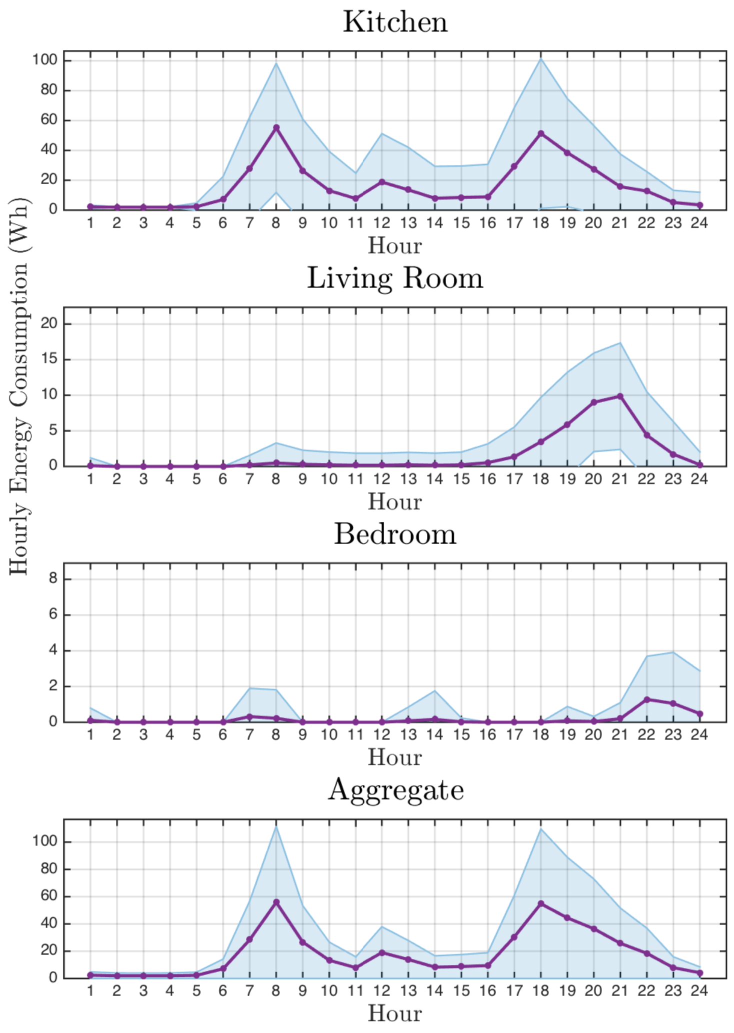


Figure 6.3: Hourly usage patterns of main lighting appliances. Solid curve represents average consumption and shaded area represents one standard deviation above mean

training data. The maximum hourly consumptions occurring in the training data are not ideal bounds to use as soft constraints, since they tend to be trivially satisfied by the improviser. Instead, for each hour h we imposed a tighter bound Δ^h_{max} on the aggregate power consumption during that hour, where Δ^h_{max} was one standard deviation above the mean consumption in hour h in the training data. Note that 89.2% of the training samples were within this bound. The values Δ^h_{max} are plotted as the shaded curve at the bottom of Figure 6.3.

To compute the probability of satisfying these constraints, we used the PRISM model checker [72]. As detailed in Section 6.3.5, the EDHMM and a monitor tracking hourly power consumption can be written as a discrete-time Markov chain. This description translates more or less directly into the PRISM modeling language. Having done this, the soft constraints can be put directly into PRISM using the PCTL formulation explained in Section 6.3.4 to obtain the hourly satisfaction probabilities $1 - \boldsymbol{\epsilon}_i,\ i = 1, \ldots, 24$.

6.4.1.3 Scenario-Based Model Calibration

As mentioned above, we tested three types of improvisers:

- **Scenario I: Uncalibrated Improviser.** This improviser uses the learned EDHMM with no model calibration.
- **Scenario II: Duration-Calibrated Improviser.** This improviser uses the duration calibration heuristic described in Section 6.3.6. From the aggregate power profile given in Figure 6.3, we identified peak power consumption as occurring during hours 7, 8, 9, 17, 18, 19, 20, and 21. For these hours, the probabilities of event durations greater than 60 minutes were set to zero and the distributions re-normalized. Figure 6.4 shows a sample set of original and calibrated event duration distributions for the 19th hour of the day.
- **Scenario III: Transition-Calibrated Improviser.** This improviser extends the previous one by also applying the transition calibration heuristic described in Section 6.3.6. The set of hours for which transition probabilities were calibrated includes the peak hours considered in the previous section, with the addition of hours 4 and 5, for which very few events were recorded in the training data. As Figure 6.3 indicates, the significant sources of power consumption are the kitchen and the living room lighting appliances. Therefore, we choose x_r to include states K, L, KL, and KLB (see Table 6.1 for label descriptions).

  Figure 6.5 depicts some hourly transition probability matrices before and after calibration. Each circle indicates a nonzero transition probability from state x_t to x_{t+1}, where its area is proportional to the probability. The blue circles show the original learned probabilities, and the green circles show the probabilities decreased by calibration. For clarity, we do not show the corresponding increases in the probabilities of transitioning to the OFF state.

6.4.2 Experimental Results

Our focus in this section is to evaluate the performance of synthesized improvisers using probabilistic model checking and to compare them based on their fidelity to soft constraints. It is also of interest to study the power profile characteristics of improvised traces to ensure scenario-based calibrations do not impact the similarity of improvisations to recorded data based on human actuation.

Figure 6.6 summarizes model checking results for the original EDHMM and for the two calibrated models with constrained power consumption properties. We

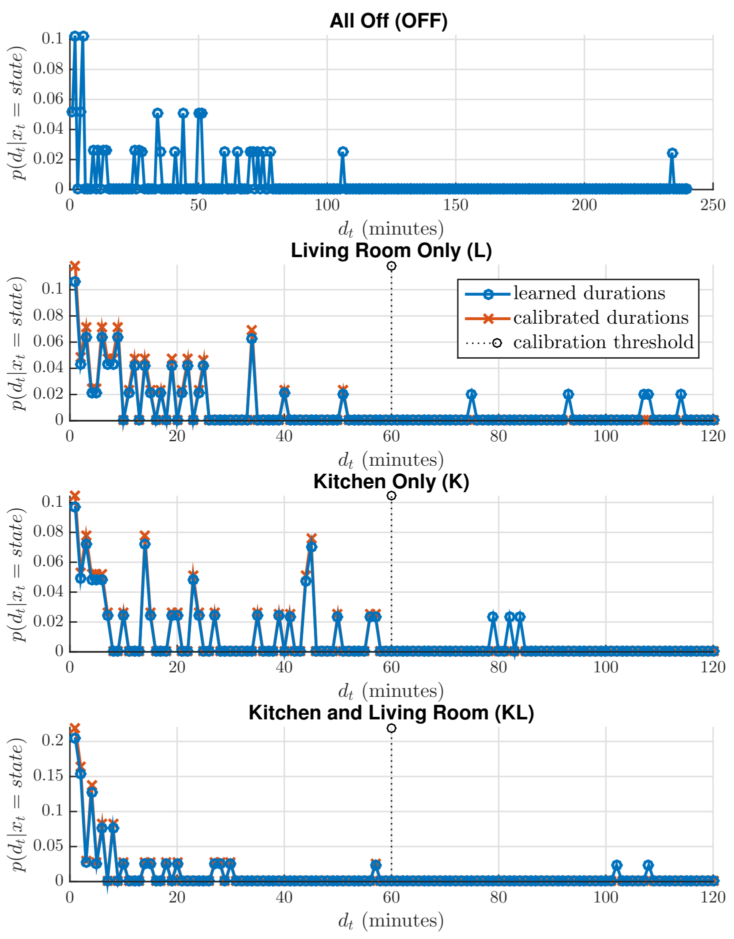


Figure 6.4: Sample learned and calibrated duration distributions for h=19

additionally provide the empirical probability of soft constraints being satisfied by training data, mainly for visual comparison. Model checking results suggest that the improviser based on the learned EDHMM behaves comparably to the empirical satisfaction probabilities, however, since the soft constraints are not explicitly enforced by the EDHMM, some hourly probabilities significantly deviate from empirical values. When we investigate model checking results for the two calibrated improvisers, which aim to improve the probability of satisfying soft constraints, we observe that the transition-calibrated improviser yields highest satisfaction probabilities on the soft constraints for all hours of the day. The duration-constrained improviser performs better than the learned model, for all hours except for hours 9, 21 and 22. As explained in Section 6.3.6, the duration heuristic does not guarantee an improvement in the probability of satisfying the soft constraints. This can be explained in this particular case by the phenomenon that at these particular hours, the state transition matrix tends to make transitions to high-consumption states more probable, and skewing the duration distribution towards zero causes more state transitions to be made during peak hours.

Figure 6.7 compares the aggregate hourly power consumption profiles obtained from the training data, with ones obtained from 100 20-day long improvisations generated by a particular lighting improviser. For all three improviser profiles, the hourly mean power trend matches that of the original data. Moreover, for calibrated im-

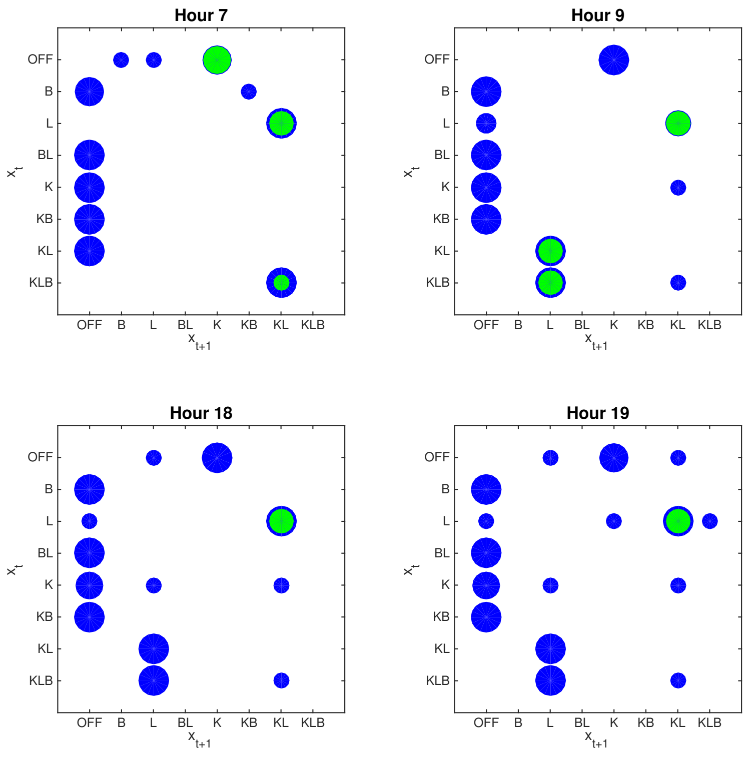


Figure 6.5: Learned vs. calibrated state transition probabilities for selected hours. (Blue: Learned A_h, Green: Probabilities adjusted by Scenario III, See Table 6.1 for label descriptions)

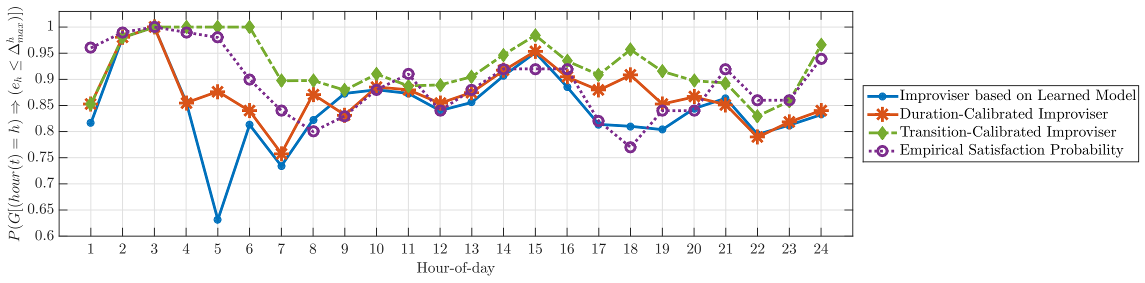


Figure 6.6: Model checking results on the satisfaction probabilities of hourly soft constraints

provisers, the one standard deviation curve above mean mostly remains within the same bound for the original data. Even though the duration-calibrated improviser has eliminated most of the highly variable power consumption trend exhibited by the uncalibrated improviser, it still demonstrates high variability in power for the hour range $h = 9, \dots, 12$ compared to the training data. This behavior is successfully mitigated by the transition-calibrated improviser, which is shown to satisfy the one sigma power constraint more strictly than the duration-calibrated improviser as expected.

Finally, in Figure 6.8, we show several day-long traces from the three improvisers together with time-aligned excerpts from the training data. Note that the uncalibrated improvisations are visually quite similar to the training data, illustrating the quality of the EDHMM as a model. The calibrated improvisations are also qualitatively similar to the training data, but somewhat sparser as we would expect from enforcing constraints on power consumption. This demonstrates how our model calibration techniques are effective at enforcing soft constraints without drastically changing system behavior.

Overall, experimental results suggest that, given a suitable learning model, it is possible to synthesize a control improviser, which produces randomized control

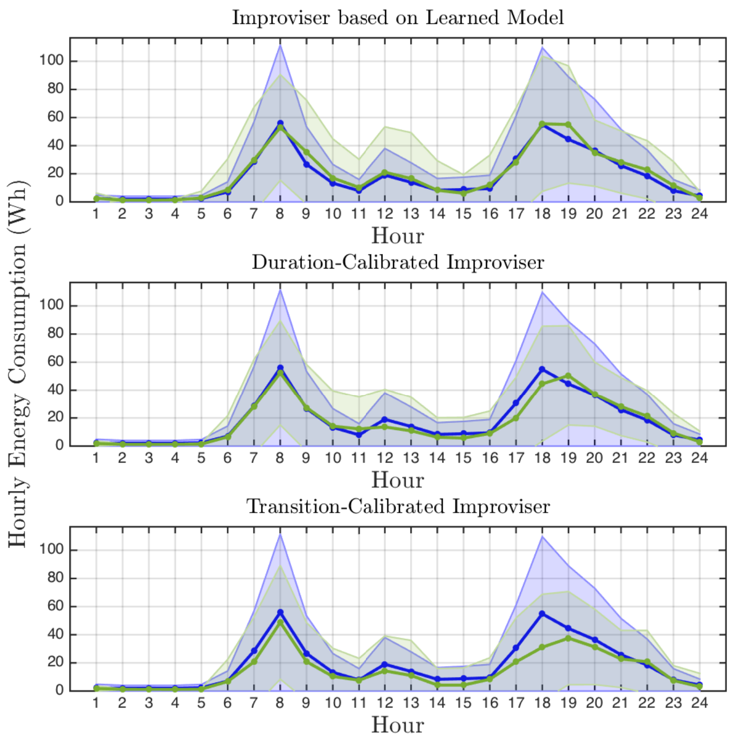


Figure 6.7: Comparison of aggregate hourly energy profiles (Blue: Training data, Green: Improvisations. Solid curves represent mean energy, shaded region represents one standard deviation from mean)

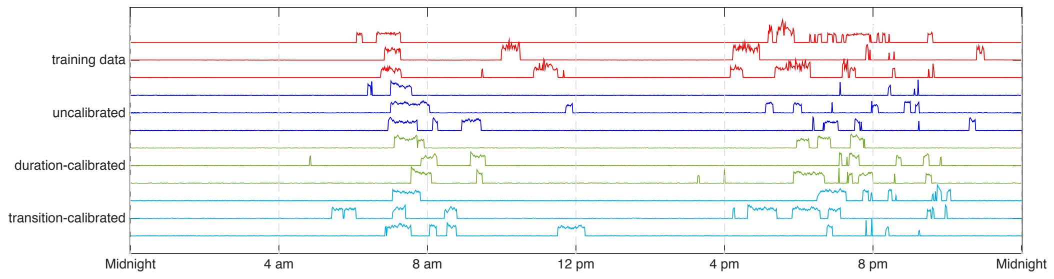


Figure 6.8: Appliance 1 activation patterns over day-long intervals

sequences that are faithful to observed system behavior. More importantly, scenario-based model calibration methods can be applied to systematically constrain the nature of randomness, which is quantifiable via probabilistic model checking. Our experiments have shown significant improvements on the satisfaction probabilities of soft constraints after applying heuristic calibrations, while preserving desired qualitative characteristics in improvised control sequences.

6.5 Related Work

Control improvisation is an automata-theoretic problem that was formally defined and analyzed in [51]. CI was applied to machine improvisation of music in [39], where a symbolic reference melody was used to synthesize an automaton that was composed with a specification automaton (capturing user-specified musical properties) to produce a control improviser. In this work, we consider a case study in the field of home automation, which enables us to learn a more general Bayesian model. We represent training data by modeling temporal progression and the stochastic characteristics of underlying events given noisy sensor data. Moreover, as an extension of our previous work, we learn specifications from user-generated data directly, and perform scenario-based calibrations on the learned model to enforce formal statistical properties.

Appliance modeling in residential settings has several proposed benefits, including reduced power consumption, automated actuation of smart appliances subject to energy pricing, microgrid load balancing, and home security [108]. Additionally, personalized advisory tools have gained popularity to provide adaptive demand-response prediction [7, 85]. Bayesian modeling techniques for home appliance load modeling has been an emerging topic of interest [65], and EDHMM-based models have previously been proposed for load disaggregation [58]. Markov modeling of uncertainties in demand and energy pricing has been studied in [92], which presents a reinforcement learning based approach to optimal load scheduling.

The related subjects of data-driven occupancy prediction [16] and user behavior modeling for energy demand predictions have also been studied in recent years. In [118], a stochastic model to predict time-dependent user activity was presented.

In contrast to these studies, we focused on building appliance models that require no prior modeling of appliances or their users, but build solely on streaming sensor data from home appliances. To achieve this goal, instead of building a context model for user behavior, we considered a context-agnostic Bayesian model that captures time-dependent appliance duration and power consumption distributions, as well as time-dependent transition probabilities for appliance activation. Integration of suitable occupancy and user prediction techniques with ours is a clear direction for future work.

Our approach in this application shares the vision of data-driven modeling of human-actuated systems, however, instead of synthesizing a Markov model of an underlying human behavior, we are interested in capturing the temporal progression of distributions associated with a semi-Markov pattern, as captured by the EDHMM. Our control objectives also vary, in that, we are interested in synthesizing a *randomized* controller that captures the probabilistic nature of a learned set of behaviors, as opposed to synthesizing optimal control traces based on hard specifications.

6.6 Conclusion

In this chapter, we address the problem of randomized control for IoT systems, with a particular focus on systems whose components can be controlled either by humans or automatically. From streams of time-stamped system events, we learn models that are assumed to vary as a function of an underlying state space governed by events with durations. We leverage the recently-proposed technique of control improvisation [51, 50] to generate randomized control sequences, which are *similar* to an observed set of behaviors, and moreover, always satisfy some desired *hard* constraints and *mostly* satisfy *soft* constraints, while exhibiting *variability*. We presented an implementation of the end-to-end control improvisation workflow using the PRISM tool to enforce soft constraints on the improviser. We evaluated our technique in the domain of home appliance control by synthesizing improvisers to control a group of lighting appliances based on learned usage patterns and subject to probabilistic constraints on power consumption. The results of our experiments showed that our methods can effectively enforce soft constraints while largely maintaining qualitative and quantitative properties of the original system's behavior.

For future work, we plan to investigate new applications of this framework in the IoT space. We also plan to investigate techniques to improve the efficiency of our scheme, as well as its implementation on real hardware.

Chapter 7

Other Application Areas of AOM

The paradigm of Aspect-Oriented Modeling (AOM) is introduced in Chapter 3, where we describe the semantics of aspects, introducing the implementation of aspects for modeling communication and network fabrics between components. In this chapter, we take the discussion one step further by presenting a set of additional essential modeling concerns addressed by aspect in the scope of cyber-physical system design.

7.1 Execution

In many embedded control applications, execution time of software influences the behavior of the application. Design space exploration is necessary to evaluate the behavior of different implementations. To this day, tool support for this activity is very limited. In this next example, we illustrate how one could build simple models of CPUs as *advices*, and associate actor executions with these advices using AOM.

In cooperative control applications, an execution bottleneck is often caused by on-line control algorithms. In the running example, it would be of interest to model execution times for the two controller models to ensure that application requirements are met at runtime. As an example, suppose we want to evaluate two alternative architectural designs. In the model depicted in Figure 7.1, two execution advices are implemented as composite actors. In the figure, the two advices represent two alternative architectures, one with a single core and one with two cores. In the figure, the `1Processor` advice is enabled on the `Observation-Optimizing Controller` and the `Main Robot Control` actors. The 1-processor model merges incoming execution requests and schedules them on a server that delays the actors for a specified amount of time, emulating execution time. A more elaborate model could include a scheduling policy such as EDF. In addition to the *enable* flag, an *execution time* parameter is added to all join points by an execution advice. As a result, every actor can be simulated to have a different execution time. As this parameter is read every time

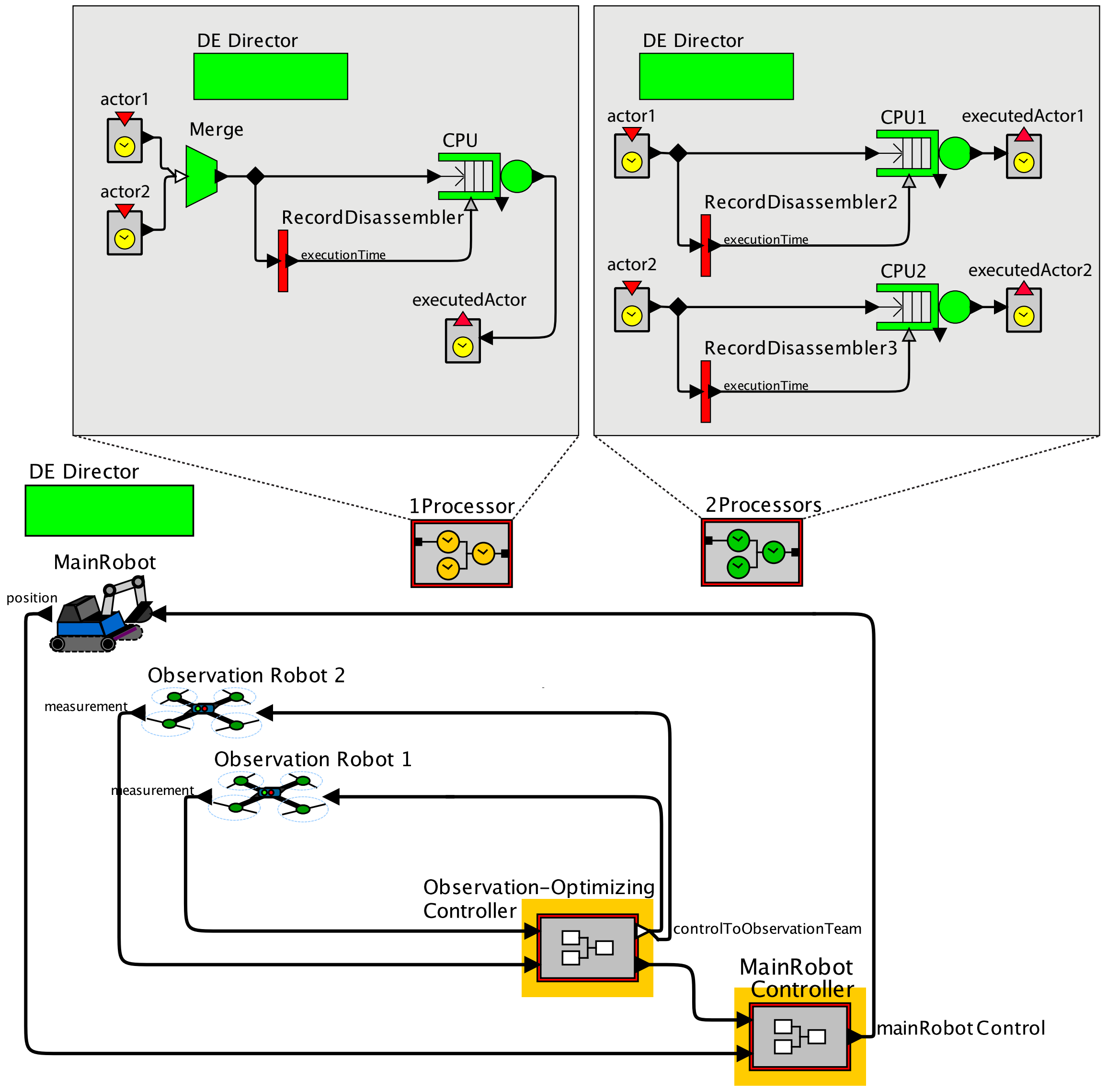


Figure 7.1: Mapping of functional models onto architecture models using aspects

the actor is executed, it is possible to provide execution times that differ in between iterations. The join points are also extended with an *execution request port*, which is a special actor inside the advice that receives *execution request tokens* when the actor is scheduled to be fired. In the example, these execution request ports are `actor1` and `actor2`. The execution request token contains an object reference of the actor to be fired and the execution time for the current firing. Before firing an actor that is decorated with a processor advice, the director triggers a firing of the advice and generates an execution request token for the actor in the appropriate execution request port.

Figure 7.2 shows an execution of the main robot controller and the observation-optimizing controller in the single and two processor cases, respectively. The illustrated execution monitor is part of the advice implementation. The execution simulations are obtained with a global sampling period of 250 ms, and worst-case execution times for `Observation-OptimizingController` and `MainRobotController` set to 250 and 50 ms, respectively. It is seen that this particular execution model is not schedulable on the single core architecture for the given execution time and sampling period parameters.

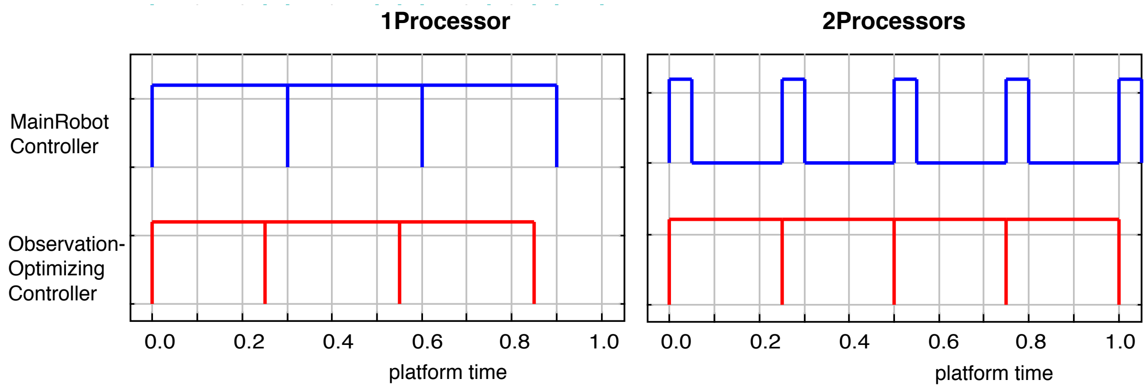


Figure 7.2: Controller execution times on different architectures

To keep the illustration simple, the model shown here is naïve. Execution times of the components are difficult to know precisely, and may need to be modeled probabilistically or to rely on sophisticated program analysis tools. Similarly, models of scheduling policies can get quite sophisticated, and the effects of resource contention in the processor architecture can get complex. But because of the effective separation of concerns, an expert on modeling and simulation of real-time software systems can focus on the design of the aspect model, while the expert on machine learning and optimization focuses on the control design. Very little coordination is required between the two.

To further demonstrate the interaction of an enabled execution aspect with the functional model, consider Figure 7.3, which illustrates the impact of a slow processor

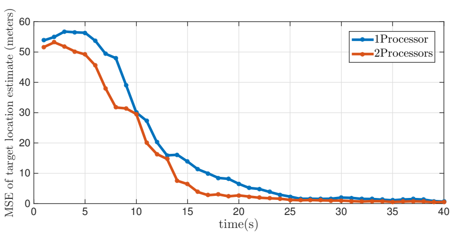


Figure 7.3: Effect of processor architecture on target estimation timing and functionality

on the target-estimation accuracy, as obtained by the Observation-Optimizing Controller. Although information from the observation team is still processed in order, the processing delay causes a slower convergence to the true target state, impacting the real-time performance, which, by an execution aspect, has been detected before deployment.

7.2 Fault Modeling

Fault models are among the most natural aspects in a multi-view system. By definition, faults are not part of the system specification itself. An orthogonal modeling paradigm helps fault models to be integrated with the system design in a way that preserves separation of concerns. In many standardized architecture description languages, error models are part of the native system specification. For instance, AADL features an error annex describing numerous fault models [114].

Faults are orthogonal concerns that can be modeled as aspects. Figure 7.4 demonstrates the top-level model of our running example, which has been decorated by two fault models: (i) a stuck-at component fault model that affects the range sensor readings of one of the observation robots, (ii) a packet drop fault model that affects the controller-to-robot communication for one of the observation robots.

Figure 7.5 illustrates the implementation of the `StuckAtFault` aspect shown to affect the output values of the `RangeSensor` component. A stuck-at fault occurs when the individual signals and pins are stuck at a fixed value on an integrated circuit. As an abstraction, such faults can be modeled to occur on a single input or output of a component. With the addition of this aspect to the functional model of `Observation Robot 2`, the sensor output values produced on board this robot will get stuck at a fixed value with some probability during execution.

A packet-drop fault, as the name suggests, models a dropped packet over a network link, often mimicking a packet erasure channel with an assigned packet erasure probability, applying independently to each transmitted packet. The `PacketDropFault` aspect that affects transmission of control inputs to `Observation Robot 2` is shown

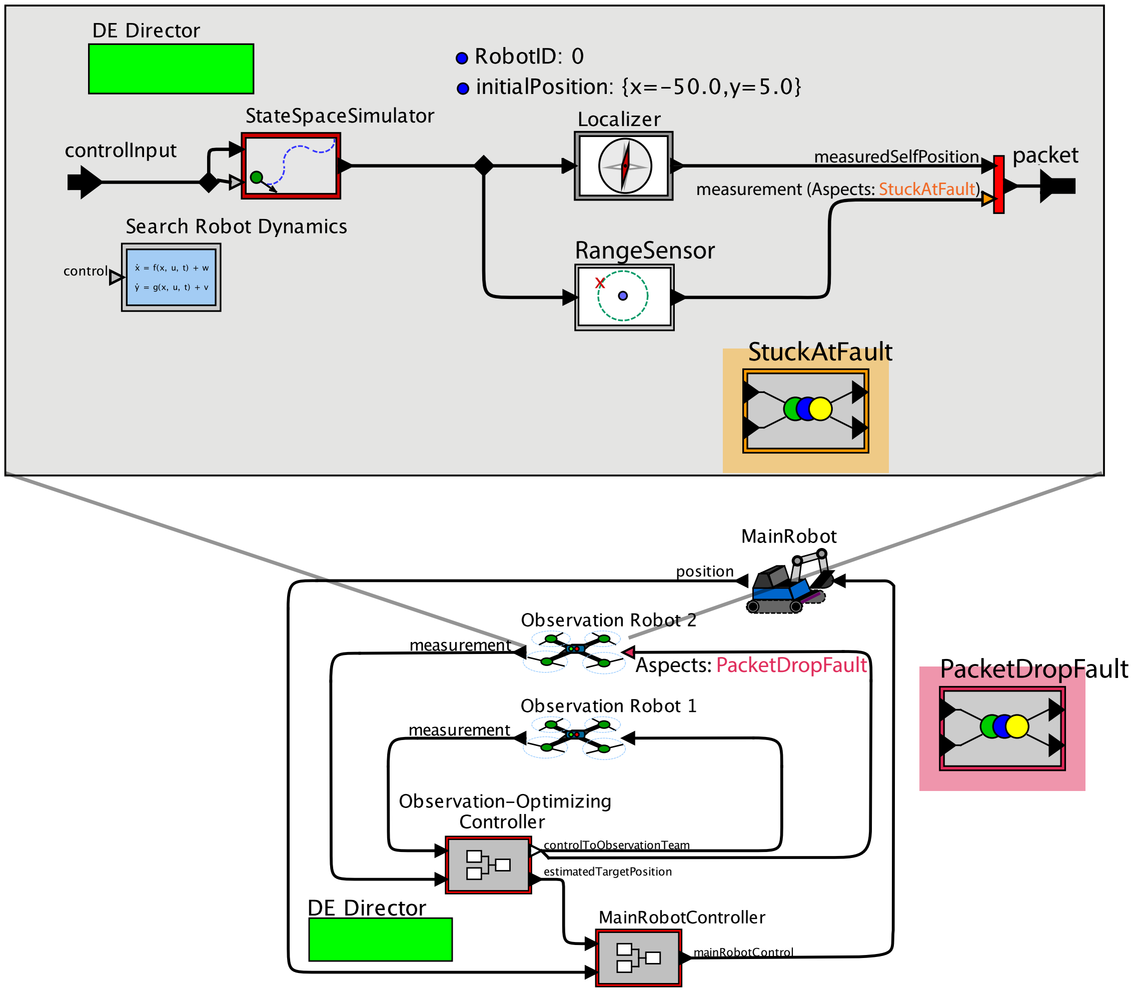


Figure 7.4: Aspect-oriented component and communication fault models

in Figure 7.6.

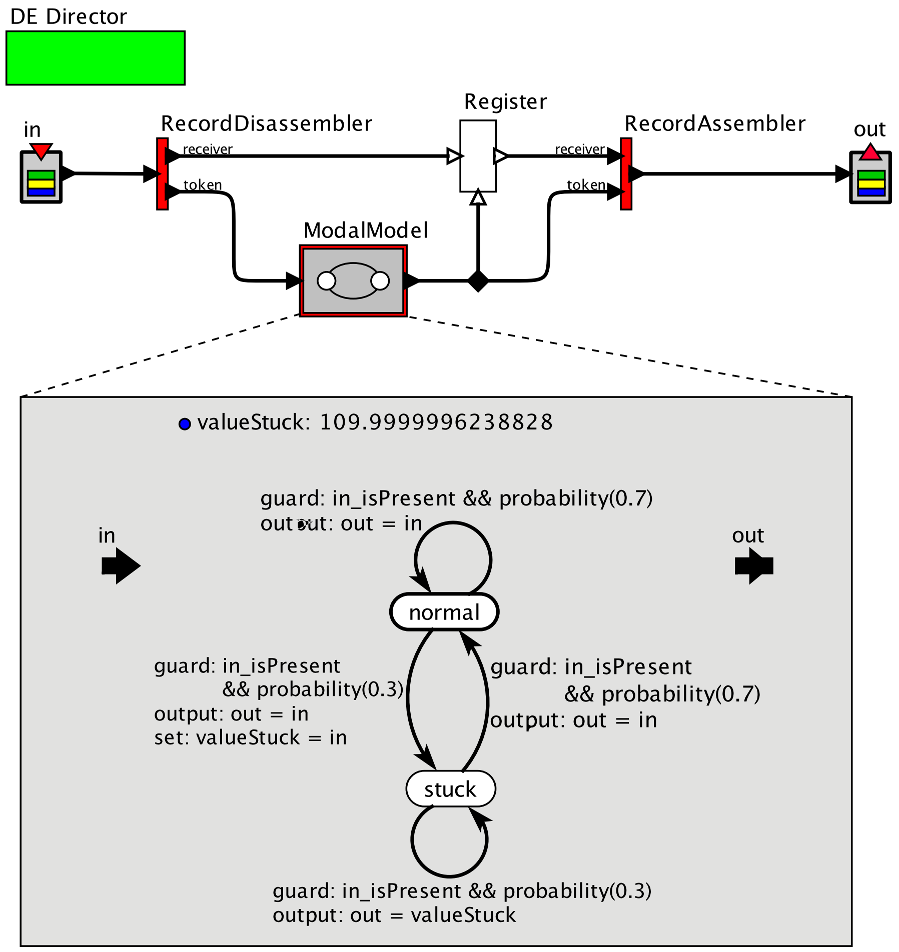


Figure 7.5: Fault model of a stuck signal.

7.3 Fault and error handling

Errors can occur due to software and hardware failures. While most software errors are eliminated through validation and verification, hardware errors cannot be eliminated easily, thus precautions have to be taken. Error handling code must be inserted to catch possible failure situations. Because errors can occur in many different places, mixing error handling functionality with the system functionality can make the model difficult to understand. Also, error handling strategies might depend on the operating conditions, and therefore might need to exhibit modal behavior.

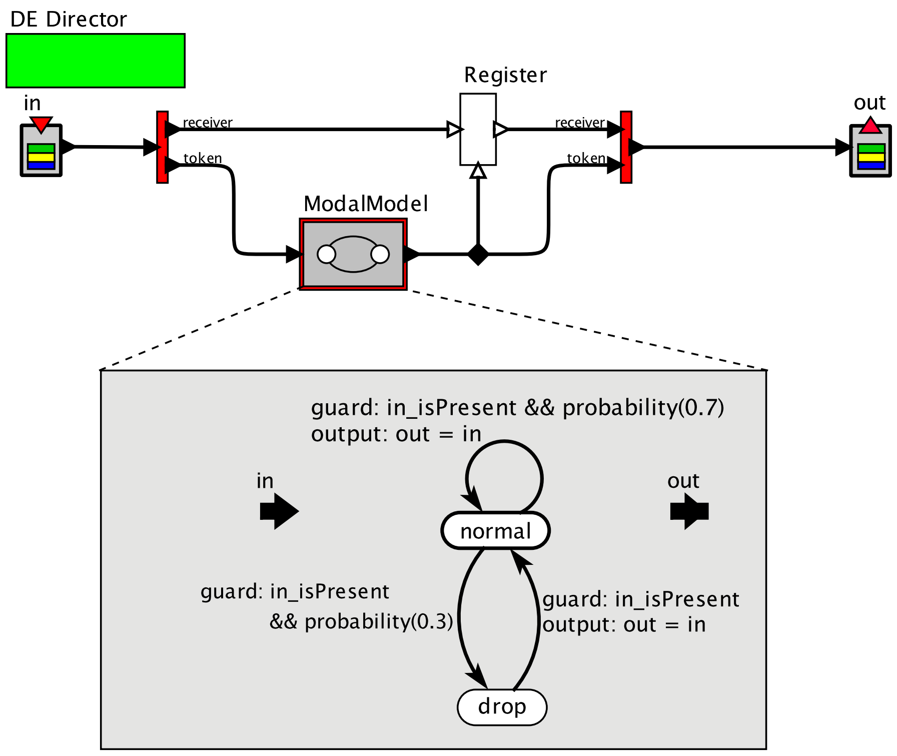


Figure 7.6: Packet Drop Fault model that drops packets with a probability.

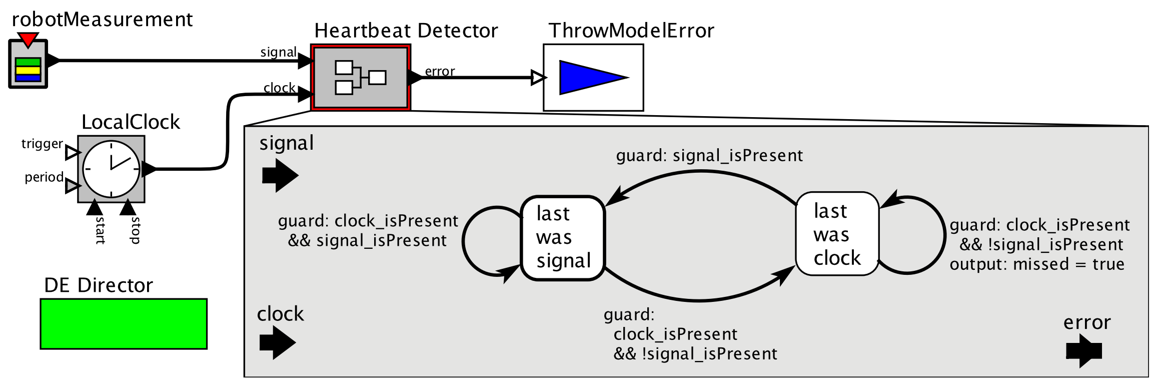


Figure 7.7: Aspect-oriented heartbeat detection

Figure 7.7 shows a model of a heartbeat detector which implements a mechanism for detecting a missed sensor reading. It uses a state machine that expects time stamped sensor readings at its input. In our running example, this input is connected to the `robotMeasurements` port. The *clock* input reads time stamped inputs from a local clock. The state machine keeps track of whether the most recent event was a sensor message or a message from the local clock. It issues a missed event if two consecutive local clock messages were received. See also [41] for a use of the HeartBeatDetector in a power plant control system. This heartbeat detector can be used on signals that come from unreliable sources.

7.4 Contract Modeling

Formal contracts [100] can be useful in CPS design to clarify interfaces and enforce documented interaction behavior between components. In such contracts, high-level system specifications are formalized as assume-guarantee formulas. Some work has been dedicated to correct-by-construction synthesis of control protocols based on temporal logic formalization of these contracts [121]. In cases where such a controller design is not possible due to complexity, when dealing with legacy systems, or to detect runtime faults that cause contract violations, one might want runtime components that monitor contract compliance. Aspect-oriented modeling enables addition of design contracts to an existing system model. If design requirements are violated in runtime, this can be detected by an aspect-oriented contract monitor.

Consider, as part of our running example, a *collision avoidance* contract defined on the main robot to ensure that the robot does not collide with a detected target or obstacle on the map. The informal contract specification is that "whenever a *proximity alert* flag has been raised, the robot must come to a full stop within X seconds, and wait for the *resolved* signal from the operator before proceeding with

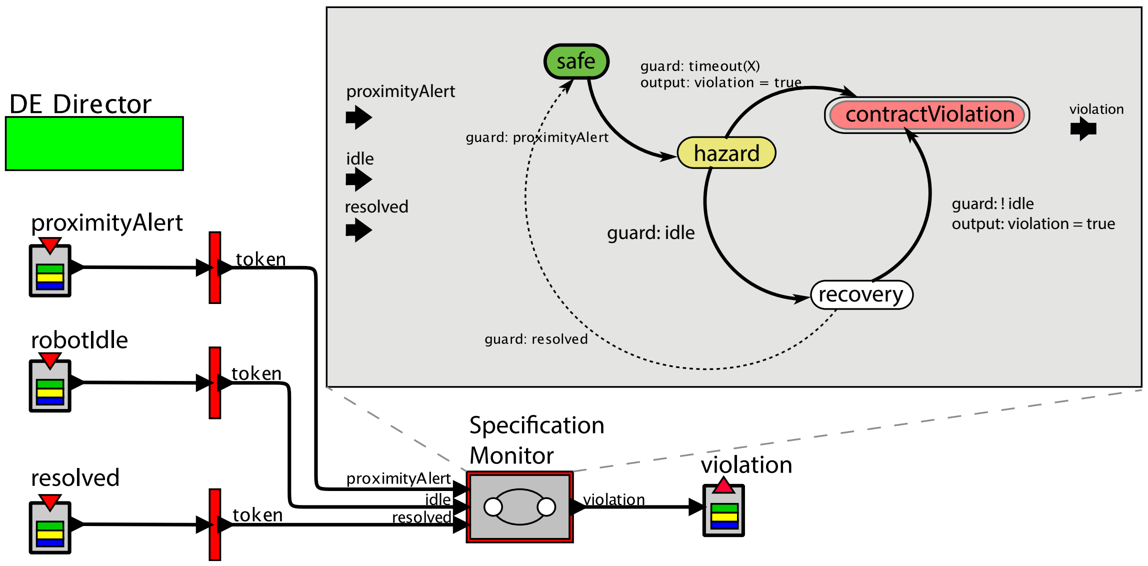


Figure 7.8: A collision avoidance contract defined as an aspect on the main robot controller

its mission."[1] This contract needs to be monitored on the the main robot controller. Figure 7.8 implements a contract monitor that, upon detecting a violation, raises a flag. This aspect can be easily and unobtrusively woven with a model of this robot system.

7.5 Logging and Debugging

Logging is a common example for aspect-oriented programming. Adding logging code to functional code or models can unnecessarily make the model more complex. Also, at different stages of the design, different information is logged. For deployment, logging is usually completely removed or disabled. Using aspects to perform logging and plotting signals is a much cleaner way to evaluate simulation results. We can modify the previously introduced network model to just log incoming messages and forward them immediately, thus implementing a message logger. An execution logger can be implemented by modifying the 1 or 2 processor models by adding a logging component and removing time delays.

With similar mechanisms as used for logging, breakpoints can be inserted into the execution of a model by using a special actor that pauses the execution (see Figure 7.9 for a potential collision-avoidance breakpoint).

[1]This contract can formally be represented by the Signal Temporal Logic (STL) formula: $G(\texttt{proximityAlert} \rightarrow F_{[0,X s]} G(\texttt{idle}\ U\ \texttt{resolved}))$.

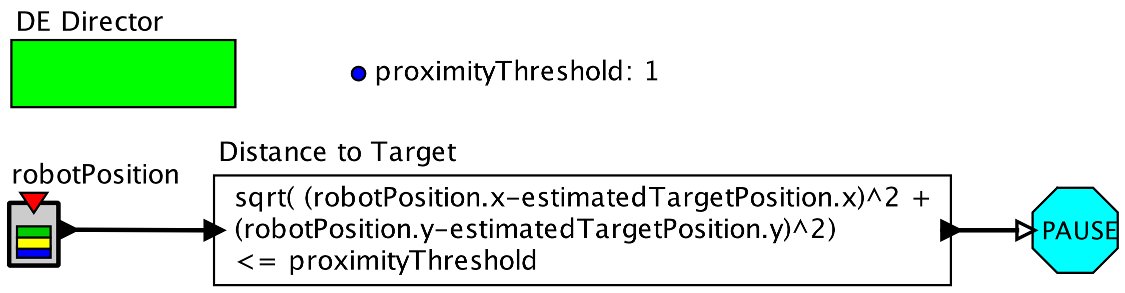

DE Director
proximityThreshold: 1
Distance to Target
robotPosition
sqrt((robotPosition.x-estimatedTargetPosition.x)^2 +
(robotPosition.y-estimatedTargetPosition.y)^2)
<= proximityThreshold
PAUSE

Figure 7.9: Model breakpoints as aspects

7.6 Conclusion

This chapter introduces application areas of the aspect-oriented modeling (AOM) paradigm, which is introduced in Chapter 3. The previously introduced application areas of modeling communication, dynamics, sensing, and environmental constraints are extended in this chapter by the discussion on modeling architecture aspects for component execution, fault modeling and detection, aspect-oriented contract modeling, and logging and debugging. While this is not an exhaustive list of the application areas of AOM, the discussed formalisms are widely applicable to other cross-cutting concerns that are relevant to developing modular CPS.

During this work, we have built up a library of aspects for common cross-cutting concerns in industrial cyber-physical system designs. These models all available for download at `http://ptolemy.org`..

Bibliography

[1] Amazon Kinesis. `https://aws.amazon.com/kinesis/`.

[2] Apache Kafka. `http://kafka.apache.org/`.

[3] Apache Samza. `http://samza.apache.org/`.

[4] Apache Spark Streaming Programming Guide. `http://spark.apache.org/docs/latest/streaming-programming-guide.html`.

[5] Apache Storm. `https://storm.apache.org/`.

[6] Balancing and frequency control. `http://www.nerc.com/docs/oc/rs/NERC%20Balancing%20and%20Frequency%20Control%20040520111.pdf`.

[7] DR-Advisor: Data-Driven Demand Response Recommender System. `https://mlab.seas.upenn.edu/dr-advisor/`.

[8] Microsoft Azure. `https://azure.microsoft.com/en-us/services/stream-analytics/`.

[9] NI LabVIEW System Design Software. `http://www.ni.com/labview/`.

[10] Node.js. `https://nodejs.org`.

[11] NSF Cyber-Physical Systems Program. `https://www.nsf.gov/funding/pgm_summ.jsp?pims_id=503286`.

[12] Particle Filtering for Tracking and Localization. `http://www.cc.gatech.edu/~dellaert/07F-Robotics/Schedule_files/11-ParticleFilters.ppt.pdf`.

[13] Robotics System Toolbox: Design and test algorithms for robotics applications. `https://www.mathworks.com/products/robotics/`.

[14] Simulink. `http://www.mathworks.com/products/simulink/`.

[15] Vert.x. `https://vertx.io`.

[16] D. Aerts, J. Minnen, I. Glorieux, I. Wouters, and F. Descamps. A method for the identification and modelling of realistic domestic occupancy sequences for building energy demand simulations and peer comparison. *Building and environment*, 75:67–78, 2014.

[17] G. Agha. Concurrent object-oriented programming. *Communications of the ACM*, 33(9), 1990.

[18] B. Akesson, A. Molnos, A. Hansson, J. A. Angelo, and K. Goossens. Composability and predictability for independent application development, verification, and execution. In M. Hübner and J. Becker, editors, *Multiprocessor System-on-Chip: Hardware Design and Tool Integration*. Springer, 2011.

[19] I. Akkaya, P. Derler, S. Emoto, and E. A. Lee. Systems engineering for industrial cyber-physical systems using aspects. *Proceedings of the IEEE*, 104(5):997–1012, 2016.

[20] I. Akkaya, S. Emoto, and E. A. Lee. PILOT: An Actor-oriented Learning and Optimization Toolkit for Robotic Swarm Applications. In *Second International Workshop on Robotic Sensor Networks, part of CPSWeek (RSN'15)*. ACM, 2015.

[21] I. Akkaya, D. J. Fremont, R. Valle, A. Donzé, E. A. Lee, and S. A. Seshia. Control improvisation with probabilistic temporal specifications. In *IEEE Conference on Internet of Things Design and Implementation*. IEEE, 2016.

[22] I. Akkaya, Y. Liu, and I. Gorton. Modeling and analysis of middleware design for streaming power grid applications. In *Proceedings of the Industrial Track of the 13th ACM/IFIP/USENIX International Middleware Conference*, page 1. ACM, 2012.

[23] I. Akkaya, Y. Liu, and E. A. Lee. Modeling and simulation of network aspects for distributed cyber-physical energy systems. In *Cyber Physical Systems Approach to Smart Electric Power Grid*. Springer, 2015.

[24] M. P. Andersen, G. Fierro, and D. E. Culler. Enabling Synergy in IoT: Platform to Service and Beyond. In *2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pages 1–12. IEEE, 2016.

[25] G. Assayag and S. Dubnov. Using factor oracles for machine improvisation. *Soft Computing*, 8(9):604–610, 2004.

[26] C. Baier and J.-P. Katoen. *Principles of model checking*. MIT Press, 2008.

[27] F. Balarin, H. Hsieh, L. Lavagno, C. Passerone, A. L. Sangiovanni-Vincentelli, and Y. Watanabe. Metropolis: an integrated electronic system design environment. *Computer*, 36(4), 2003.

[28] N. Bezzo, J. Park, A. King, P. Gebhard, R. Ivanov, and I. Lee. Poster abstract:ROSLab-A modular programming environment for robotic applications. In *Cyber-Physical Systems (ICCPS), 2014 ACM/IEEE International Conference on*, pages 214–214. IEEE, 2014.

[29] G. E. Box and D. A. Pierce. Distribution of residual autocorrelations in autoregressive-integrated moving average time series models. *Journal of the American statistical Association*, 65(332):1509–1526, 1970.

[30] E. Bringmann and A. Krämer. Model-based testing of automotive systems. In *2008 1st International Conference on Software Testing, Verification, and Validation*, pages 485–493. IEEE, 2008.

[31] Y. U. Cao, A. S. Fukunaga, and A. Kahng. Cooperative mobile robotics: Antecedents and directions. *Autonomous robots*, 4(1), 1997.

[32] B. Charrow, V. Kumar, and N. Michael. Approximate Representations for Multi-Robot Control Policies that Maximize Mutual Information. In *Proc. Robotics: Science and Systems Conference (RSS)*, 2013.

[33] B. Charrow, N. Michael, and V. Kumar. Cooperative multi-robot estimation and control for radio source localization. *The International Journal of Robotics Research*, 33(4):569–580, 2014.

[34] J. E. Dagle. Postmortem analysis of power grid blackouts - the role of measurement systems. *IEEE Power and Energy Magazine*, 4(5):30–35, Sept 2006.

[35] O.-J. Dahl and K. Nygaard. Simula-a language for programming and description of descrete event systems. 1965.

[36] D. de Niz and P. H. Feiler. Aspects in the industry standard AADL. In *Proceedings of the 10th international workshop on Aspect-oriented modeling*, AOM '07, New York, NY, USA, 2007. ACM.

[37] B. Denis, J. Keignart, and N. Daniele. Impact of nlos propagation upon ranging precision in uwb systems. In *Ultra Wideband Systems and Technologies, 2003 IEEE Conference on*, pages 379–383. IEEE, 2003.

[38] P. Derler, E. A. Lee, and A. S. Vincentelli. Modeling cyber–physical systems. *Proceedings of the IEEE*, 100(1):13–28, 2012.

[39] A. Donzé, R. Valle, I. Akkaya, S. Libkind, S. A. Seshia, and D. Wessel. Machine improvisation with formal specifications. In *Proceedings of the 40th International Computer Music Conference (ICMC)*, 2014.

[40] A. Doucet and A. M. Johansen. A tutorial on particle filtering and smoothing: Fifteen years later. *Handbook of Nonlinear Filtering*, 12:656–704, 2009.

[41] J. Eidson, E. A. Lee, S. Matic, S. A. Seshia, and J. Zou. A time-centric model for cyber-physical applications. In *Proceedings of 3rd International Workshop on Model Based Architecting and Construction of Embedded System (ACESMB 2010)*, pages 21–35, October 2010.

[42] S. Emoto, I. Akkaya, and E. A. Lee. Information seeking and model predictive control of a cooperative robot swarm. *Proc. SWARM*, 2015.

[43] S. Emoto, I. Akkaya, and E. A. Lee. Information seeking and model predictive control of a cooperative multi-robot system. *Artificial Life and Robotics*, pages 1–6, 2016.

[44] H. Espinoza, H. Dubois, S. Gérard, J. Medina, D. C. Petriu, and M. Woodside. Annotating UML models with non-functional properties for quantitative analysis. In *MoDELS*, volume LNCS 3844 of *Lecture Notes in Computer Science*. Springer-Verlag, 2005.

[45] P. H. Feiler and D. P. Gluch. *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language*. Addison-Wesley Professional, 1st edition, 2012.

[46] F. Ferrari, R. Burrows, O. Lemos, A. Garcia, E. Figueiredo, N. Cacho, F. Lopes, N. Temudo, L. Silva, S. Soares, A. Rashid, P. Masiero, T. Batista, and J. Maldonado. An exploratory study of fault-proneness in evolving aspect-oriented programs. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 65–74, New York, NY, USA, 2010. ACM.

[47] L. Filipponi, A. Vitaletti, G. Landi, V. Memeo, G. Laura, and P. Pucci. Smart city: An event driven architecture for monitoring public spaces with heterogeneous sensors. In *Sensor Technologies and Applications (SENSORCOMM), 2010 Fourth International Conference on*, pages 281–286, July 2010.

[48] A. Fisher, C. Jacobson, E. A. Lee, R. Murray, A. Sangiovanni-Vincentelli, and E. Scholte. Industrial cyber-physical systems — iCyPhy. In *Complex Systems Design & Management (CSD&M)*, Paris, France, 2013. Springer.

[49] I. Forrester. The Forrester Wave: Big Data Streaming Analytics Platforms, Q3 2014, 2014.

[50] D. J. Fremont, A. Donzé, S. A. Seshia, and D. Wessel. Control improvisation. *arXiv preprint arXiv:1411.0698*, 2014.

[51] D. J. Fremont, A. Donzé, S. A. Seshia, and D. Wessel. Control Improvisation. In *35th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015)*, pages 463–474, 2015.

[52] E. Galceran, A. G. Cunningham, R. M. Eustice, and E. Olson. Multipolicy decision-making for autonomous driving via changepoint-based behavior prediction. In *Proc. Robot.: Sci. & Syst. Conf*, number 1, page 2, 2015.

[53] C. Gamarra and J. M. Guerrero. Computational optimization techniques applied to microgrids planning: a review. *Renewable and Sustainable Energy Reviews*, 48:413–424, 2015.

[54] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[55] C. Gomez and J. Paradells. Wireless home automation networks: A survey of architectures and technologies. *IEEE Communications Magazine*, 48(6):92–101, 2010.

[56] J. D. Gradecki and N. Lesiecki. *Mastering AspectJ: aspect-oriented programming in Java*. John Wiley & Sons, 2003.

[57] J. Gray, T. Bapty, S. Neema, D. C. Schmidt, A. Gokhale, and B. Natarajan. An approach for supporting aspect-oriented domain modeling. In *Proceedings of the 2nd International Conference on Generative Programming and Component Engineering*, GPCE '03, New York, NY, USA, 2003. Springer-Verlag New York, Inc.

[58] Z. Guo, Z. J. Wang, and A. Kashani. Home appliance load modeling from aggregated smart meter data. *Power Systems, IEEE Transactions on*, 30(1):254–262, 2015.

[59] B. Hartmann, S. R. Klemmer, M. Bernstein, L. Abdulla, B. Burr, A. Robinson-Mosher, and J. Gee. Reflective physical prototyping through integrated design, test, and analysis. In *Proc. of User interface software and technology*, pages 299–308. ACM, 2006.

[60] C. Hewitt. Viewing control structures as patterns of passing messages. *Journal of Artificial Intelligence*, 8(3), 1977.

[61] G. M. Hoffmann and C. J. Tomlin. Mobile sensor network control using mutual information methods and particle filters. *Automatic Control, IEEE Transactions on*, 55(1):32–47, 2010.

[62] J. Holt and S. Perry. *SysML for systems engineering*, volume 7. IET, 2008.

[63] M. F. Huber, T. Bailey, H. Durrant-Whyte, and U. D. Hanebeck. On entropy approximation for Gaussian mixture random vectors. In *Multisensor Fusion and Integration for Intelligent Systems, 2008*, pages 181–188. IEEE, 2008.

[64] R. Johnson, J. Hoeller, A. Arendsen, T. Risberg, and C. Sampaleanu. *Professional Java Development with the Spring Framework*. Wiley, 2005.

[65] Z. Kang, M. Jin, and C. Spanos. Modeling of end-use energy profile: An appliance-data-driven stochastic approach. In *Industrial Electronics Society, IECON 2014 - 40th Annual Conference of the IEEE*, pages 5382–5388, Oct 2014.

[66] G. Karsai, A. Lang, and S. Neema. Design patterns for open tool integration. *Software and Systems Modeling*, 4(2), 2005.

[67] G. Karsai, J. Sztipanovits, A. Ledeczi, and T. Bapty. Model-integrated development of embedded software. *Proceedings of the IEEE*, 91(1), 2003.

[68] J. Kelly and W. Knottenbelt. The UK-DALE dataset, domestic appliance-level electricity demand and whole-house demand from five UK homes. 2, 2015.

[69] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold. An overview of AspectJ. In *Proceedings of the 15th European Conference on Object-Oriented Programming*, ECOOP '01, pages 327–353, London, UK, UK, 2001. Springer-Verlag.

[70] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In *European Conference in Object-Oriented Programming (ECOOP)*, volume LNCS 1241, Finland, 1997. Springer-Verlag.

[71] A. Kushleyev, D. Mellinger, C. Powers, and V. Kumar. Towards a swarm of agile micro quadrotors. *Autonomous Robots*, 35(4):287–300, 2013.

[72] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[73] R. Laddad. *AspectJ in action: practical aspect-oriented programming.* Dreamtech Press, 2003.

[74] E. Latronico, E. A. Lee, M. Lohstroh, C. Shaver, A. Wasicek, and M. Weber. A vision of swarmlets. *IEEE Internet Computing*, 19(2):20–28, 2015.

[75] E. A. Lee. The problem with threads. *Computer*, 39(5):33–42, 2006.

[76] E. A. Lee. Cyber physical systems: Design challenges. In *International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC)*, Orlando, Florida, 2008. IEEE.

[77] E. A. Lee. Cyber physical systems: Design challenges. In *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*, pages 363–369. IEEE, 2008.

[78] E. A. Lee. The past, present and future of cyber-physical systems: A focus on models. *Sensors*, 15(3):4837–4869, 2015.

[79] E. A. Lee. IoT and CPS: A Focus on Modeling, February 2016. U.S.-German Workshop on the Internet of Things (IoT) / Cyber-Physical Systems (CPS) 19-20 January, 2016 - Washington DC, USA.

[80] E. A. Lee, J. D. Kubiatowicz, J. M. Rabaey, A. L. Sangiovanni-Vincentelli, S. A. Seshia, J. Wawrzynek, D. Blaauw, P. Dutta, K. Fu, C. Guestrin, R. Jafari, D. Jones, V. Kumar, R. Murray, G. Pappas, A. Rowe, C. M. Sechen, T. S. Rosing, B. Taskar, and D. Wessel. The TerraSwarm Research Center (TSRC) (A White Paper). Technical Report UCB/EECS-2012-207, EECS Department, University of California, Berkeley, Nov 2012.

[81] E. A. Lee, S. Neuendorffer, and M. J. Wirthlin. Actor-oriented design of embedded hardware and software systems. *Journal of Circuits, Systems, and Computers*, 12(3), 2003.

[82] E. A. Lee and A. Sangiovanni-Vincentelli. A framework for comparing models of computation. *IEEE Transactions on Computer-Aided Design of Circuits and Systems*, 17(12):1217–1229, 1998.

[83] E. A. Lee and S. A. Seshia. *Introduction to embedded systems: A cyber-physical systems approach.* Lee & Seshia, 2011.

[84] Y. Li, H. Lu, and H. Zhang. Optimistic programming of touch interaction. *ACM Trans. Comput.-Hum. Interact.*, 21(4):24:1–24:24, August 2014.

[85] Y. Li, B. L. Ng, M. Trayer, and L. Liu. Automated residential demand response: Algorithmic implications of pricing models. *Smart Grid, IEEE Transactions on*, 3(4):1712–1721, 2012.

[86] J. Liu and L. Zhang. QoS modeling for cyber-physical systems using aspect-oriented approach. In *ICNDC '11*, Washington, DC, USA, 2011. IEEE Computer Society.

[87] Y. Liu, L. Zhan, Y. Zhang, P. N. Markham, D. Zhou, J. Guo, Y. Lei, G. Kou, W. Yao, J. Chai, et al. Wide-area-measurement system development at the distribution level: An fnet/grideye example. *IEEE Transactions on Power Delivery*, 31(2):721–731, 2016.

[88] S. Loukil, S. Kallel, B. Zalila, and M. Jmaiel. AO4AADL: Aspect oriented extension for AADL. *Central European Journal of Computer Science*, 3(2):43–68, 2013.

[89] S. Mehrotra. On the implementation of a primal-dual interior point method. *SIAM Journal on optimization*, 2(4):575–601, 1992.

[90] N. Michael, S. Shen, K. Mohta, Y. Mulgaonkar, V. Kumar, K. Nagatani, Y. Okada, S. Kiribayashi, K. Otake, K. Yoshida, et al. Collaborative mapping of an earthquake-damaged building via ground and aerial robots. *Journal of Field Robotics*, 29(5):832–841, 2012.

[91] K. P. Murphy. *Dynamic bayesian networks: representation, inference and learning*. PhD thesis, University of California, Berkeley, 2002.

[92] D. O. Neill, M. Levorato, A. Goldsmith, and U. Mitra. Residential demand response using reinforcement learning. In *Smart Grid Communications, 2010 First IEEE International Conference on*, pages 409–414. IEEE, 2010.

[93] P. Nuzzo, H. Xu, N. Ozay, J. B. Finn, A. L. Sangiovanni-Vincentelli, R. M. Murray, A. Donzé, and S. A. Seshia. A contract-based methodology for aircraft electric power system design. *IEEE Access*, 2:1–25, 2014.

[94] J. Ousterhout. Why threads are a bad idea (for most purposes). In *Presentation given at the 1996 Usenix Annual Technical Conference*, volume 5. San Diego, CA, USA, 1996.

[95] L. Perea, J. How, L. Breger, and P. Elosegui. Nonlinearity in sensor fusion: Divergence issues in EKF, modified truncated SOF, and UKF. In *AIAA Guidance, Navigation and Control Conference and Exhibit*, volume 6514, 2007.

[96] C. Ptolemaeus, editor. *System Design, Modeling, and Simulation using Ptolemy II*. Ptolemy.org, Berkeley, CA, 2014.

[97] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, volume 3, page 5, 2009.

[98] L. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[99] B. Ristic, S. Arulampalam, and N. Gordon. Beyond the Kalman filter. *IEEE Aerospace and Electronic Systems Magazine*, 19(7):37–38, 2004.

[100] A. Sangiovanni-Vincentelli, W. Damm, and R. Passerone. Taming Dr. Frankenstein: Contract-Based Design for Cyber-Physical Systems. *European Journal of Control*, 2012. In press.

[101] A. Sangiovanni-Vincentelli. Quo vadis, SLD? reasoning about the trends and challenges of system level design. *Proceedings of IEEE*, 95(3), 2007.

[102] D. C. Schmidt, M. Stal, H. Rohnert, and F. Buschmann. *Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects*, volume 2. John Wiley & Sons, 2013.

[103] M. Schwager, P. Dames, D. Rus, and V. Kumar. A multi-robot control policy for information gathering in the presence of unknown hazards. In *Proc. of the Int. Symp. on Robotics Research (ISRR 11)*, 2011.

[104] J. Scott. Stream processing everywhere - What to Use. `https://www.mapr.com/blog/stream-processing-everywhere-what-use`.

[105] R. H. Shumway and D. S. Stoffer. *Time series analysis and its applications: with R examples*. Springer Science & Business Media, 2010.

[106] M. Störzer and C. Koppen. Pcdiff: Attacking the fragile pointcut problem, abstract. In *European Interactive Workshop on Aspects in Software*, Berlin, Germany, September 2004.

[107] M. Strelec, K. Macek, and A. Abate. Modeling and simulation of a microgrid as a stochastic hybrid system. In *Innovative Smart Grid Technologies (ISGT Europe), 2012 3rd IEEE PES International Conference and Exhibition on*, pages 1–9. IEEE, 2012.

[108] L. G. Swan and V. I. Ugursal. Modeling of end-use energy consumption in the residential sector: A review of modeling techniques. *Renewable and sustainable energy reviews*, 13(8):1819–1835, 2009.

[109] W. Thies, M. Karczmarek, and S. Amarasinghe. StreamIt: A language for streaming applications. In *International Conference on Compiler Construction*, pages 179–196. Springer, 2002.

[110] S. Thrun. Particle filters in robotics. In *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, pages 511–518. Morgan Kaufmann Publishers Inc., 2002.

[111] M. M. Tiller. *Introduction to Physical Modeling with Modelica.* Kluwer Academic Publishers, 2001.

[112] Tokyo Electric Power Company (TEPCO). Cleanup work by using a robot in Unit 3 Reactor Building at Fukushima Daiichi Power Station. `http://www.tepco.co.jp/en/news/library/archive-e.html?video_uuid=j18842aw&catid=61793`, July 2011.

[113] Tokyo Electric Power Company (TEPCO). An unprecedented challenge: Robots obtain crucial information on conditions inside the Fukushima Reactor. `http://www.tepco.co.jp/en/news/library/archive-e.html?video_uuid=o9fi533l&catid=69631`, April 2015.

[114] S. Vestal. An overview of the architecture analysis & design language (AADL) error model annex. In *AADL Workshop*, 2005.

[115] A. Wasicek, P. Derler, and E. A. Lee. Aspect-oriented modeling of attacks in automotive cyber-physical systems. In *Proceedings of the 51st Design Automation Conference (DAC)*, June 2014.

[116] M. Wehrmeister, C. Pereira, and F. Rammig. Aspect-oriented model-driven engineering for embedded systems applied to automation systems. *Industrial Informatics, IEEE Transactions on*, 9(4), 2013.

[117] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenstr. The worst-case execution-time problem - overview of methods and survey of tools. *ACM TECS*, 7(3), 2008.

[118] U. Wilke, F. Haldi, J.-L. Scartezzini, and D. Robinson. A bottom-up stochastic model to predict building occupants' time-dependent activities. *Building and Environment*, 60:254–264, 2013.

[119] J. L. Williams. *Information theoretic sensor management.* PhD thesis, Massachusetts Institute of Technology, 2007.

[120] M. Wimmer, A. Schauerhuber, G. Kappel, W. Retschitzegger, W. Schwinger, and E. Kapsammer. A survey on UML-based aspect-oriented design modeling. *ACM Comput. Surv.*, 43, 2011.

[121] T. Wongpiromsarn, U. Topcu, and R. M. Murray. Automatic synthesis of robust embedded control software. In *AAAI Spring Symposium: Embedded Reasoning*. AAAI, 2010.

[122] D. Wyatt. *Akka concurrency*. Artima Incorporation, 2013.

[123] H. Yin, C. Bockisch, and M. Aksit. A fine-grained debugger for aspect-oriented programming. In *Proceedings of the 11th annual international conference on Aspect-oriented Software Development*, AOSD '12, New York, NY, USA, 2012. ACM.

[124] S.-Z. Yu. Hidden semi-markov models. *Artificial Intelligence*, 174(2):215–243, 2010.

[125] H. Zeng, A. Davare, A. Sangiovanni-Vincentelli, S. Sonalkar, S. Kanajan, and C. Pinello. Design space exploration of automotive platforms in metropolis. Technical report, SAE Technical Paper, 2006.

Appendix A

Particle Filter Formulations

A.1 Inference on SSMs via Forward Computations

Consider the State-space model (SSM)

$$x_1 \sim \pi_x(x_1) \tag{A.1}$$

$$z_t|x_t \sim g(x_t, u_t, t) \tag{A.2}$$

$$x_{t+1}|x_t \sim f(x_t, u_t, t) \tag{A.3}$$

As explained in section 4, inference on SSMs is a widely encountered problem, which tries to solve for an estimate of state at time t, x_t, given all observations of the system made up to (and including) time t, denoted as $z_{1:t}$. Assuming a generic SSM and using Bayes rule, we can factor out the state estimate as

$$\begin{aligned} p(x_t|z_{1:t}) = p(x_t|z_t, z_{1:t-1}) &= \frac{p(z_t|x_t, z_{1:t-1})p(x_t|z_{1:t-1})}{p(z_t|z_{1:t-1})} \\ &= \frac{g(z_t|x_t)p(x_t|z_{1:t-1})}{p(z_t|z_{1:t-1})} \,. \end{aligned} \tag{A.4}$$

In (A.4), the posterior state distribution $p(x_t|z_{1:t})$ is factored out into a form that can be computed recursively, if we use $p(x_t|z_{1:t-1})$ obtained at the previous step. We write this term as a marginalization over the state at time $t-1$ (x_{t-1}) as:

$$\begin{aligned} p(x_t|z_{1:t-1}) &= \int p(x_t, x_{t-1}|z_{1:t-1})dx_{t-1} \\ &= \int p(x_t|x_{t-1}, z_{1:t-1})p(x_{t-1}|z_{1:t-1})dx_{t-1} \\ &= \int f(x_t|x_{t-1})p(x_{t-1}|z_{1:t-1})dx_{t-1} \end{aligned} \tag{A.5}$$

To summarize:

$$p(x_t|z_{1:t}) = \frac{\overbrace{g(z_t|x_t)}^{\text{measurement}}\ \overbrace{p(x_t|z_{1:t-1})}^{\text{prediction}}}{p(z_t|z_{1:t-1})} \qquad \text{(Measurement Update)} \tag{A.6}$$

$$p(x_t|z_{1:t-1}) = \int \underbrace{f(x_t|x_{t-1})}_{\text{dynamics}}\underbrace{p(x_{t-1}|z_{1:t-1})}_{\text{previous estimate}} dx_{t-1} \quad \text{(Prediction Step (Time Update))} \tag{A.7}$$

Note that if $p(x_{t_t}|z_{1:t-1})$ is a proposal distribution for $f(x_t|x_{t-1})$, this integral actually yields the expectation $\mathbb{E}[x_t|x_{t-1}]$, a point estimate or the state at time t.

A.1.1 T-step Prediction

$$p(x_{t+T}|z_{1:t}) = \int \prod_{l=t}^{t+T} f(x_l + 1|x_l)p(x_{t:t+T}|z_{1:t})d_{t:t+T-1} \tag{A.8}$$

$$= \int \prod_{l=t}^{t+T} f(x_l + 1|x_l)p(x_t|z_{1:t})d_{t:t+T-1} \tag{A.9}$$

A.2 The Monte-Carlo Approximation

The expectation of a random function $f(y)$ $f : \mathcal{Y} \Rightarrow \mathbb{R}^{n_f}$, where x is distributed according to $p(x)$ can be written as

$$I[f(x)] := \mathbb{E}_p[f(x)] = \int f(x)p(x)dx \tag{A.10}$$

Assume we generate i.i.d. samples from $p(\cdot)$ and represent the empirical density as:

$$\hat{p}_{MC}(x) = \frac{1}{N}\sum_{i=1}^{N}\delta_x^i(x)\,,$$

where $\delta_x^i(.)$ is the Dirac point mass located at $x \in \mathcal{X}$. Inserting this approximation into A.10 yields the expression:

$$I[f(x)] \approx \hat{I}[f(x)] = \int f(x)\frac{1}{N}\sum_{i=1}^{N}\delta_x^i(x)dx\,. \tag{A.11}$$

$$= \frac{1}{N}\sum_{i=1}^{N} f(x^i) \tag{A.12}$$

By the strong law of large numbers,

$$\hat{I}[f(x)] \xrightarrow{\text{a.s.}} I[f(x)], \quad N \to \infty$$

A.3 Importance Sampling

One problem handled by Monte Carlo methods is to approximate an expectation $\mathbb{E}_p[f(x)] = \int f(x)p(x)dx$, using samples of $p(\cdot)$. However, in many applications, $p(x)$ is generally a complex distribution, hard to directly sample from. Instead, we consider:

$$\int f(x)(\frac{p(x)}{q(x)})q(x)dx\,, \tag{A.13}$$

where $q(x)$ is easy to sample from. Define the weight function as the ratio:

$$w(x) \triangleq \frac{p(x)}{q(x)}$$

Now, consider the Monte Carlo sum sampled from $q(x)$:

$$\frac{1}{N}\sum_{i=1}^{N} f(x^{(i)})w(x^{(i)}) = \frac{1}{N}\sum_{i=1}^{N} f(x^{(i)})\frac{p(x^{(i)})}{q(x^{(i)})} \xrightarrow{a.s.} \tag{A.14}$$

$$\int f(x)\frac{p(x)}{q(x)}q(x)dx = \int f(x)p(x)dx$$

by the Strong Law of Large Numbers. Here, $q(x)$ is often referred to as an *importance density*, or equivalently, a *proposal distribution*. To avoid sampling from a complicated distribution $p(x)$, one can sample instead from a simpler $q(x)$, and compute the ratio of probabilities of the two distributions at a given sample to compute an equivalent Monte Carlo approximation.

A.4 Particle Filtering for State Estimation

Consider the State-Space model given by A.1. Estimating the posterior state distribution $p(x_t|z_{1:t})$ at time t analytically may become overly complex, especially for non-linear dynamics $f(\cdot), g(\cdot)$, and under non-Gaussian disturbances. The idea is to use a Monte Carlo sampling approach to approximate the posterior distribution, using a procedure called Sequential Importance Resampling (SIR) [40].

The operation of the particle filter is based on the alternate form of the posterior state estimate obtained by combining A.6-A.7 as:

$$p(x_t|z_{1:t}) = \frac{1}{p(z_t|z_{1:t-1})} p(z_t|x_t) \int p(x_t|x_{t-1}) p(x_{t-1}|z_{1:t-1}) d_{x_{t-1}} \tag{A.15}$$

$$= \theta g(z_t|x_t) \underbrace{\int f(x_t|x_{t-1}) p(x_{t-1}|z_{1:t-1}) d_{x_{t-1}}}_{\text{prior predictive density}} \tag{A.16}$$

Note that the first term of A.15 is only a normalization constant, the second term is the *measurement equation*, and the third term is the *dynamics equation* specified by the SSM. The final term is the posterior state distribution from the previous time step, $t-1$. The working principle of the particle filter is to recursively compute posterior state densities by using these four pieces of information. Using the importance sampling idea, where the posterior is too complex to directly sample from, the proposal density q(x) is often chosen to be the prior information (that is, the probability distribution available *before* a measurement at time t has been observed). Then, the importance weight assigned to the particle $w(t) = \frac{p(x)}{q(x)}$, would become proportional to the ratio of the measurement likelihood to the prior.

Assume that the state distribution at time t is already approximated as a mass function consisting of N particles given by $\{\tilde{x}_i^t, w_i^t\}, i = 1, 2, ..., N$, where $\tilde{x}_i^t$ is the i'th particle sampled at time t, and w_i is the associated particle weight. Then, choosing the proposal density q(x) proportional to this distribution becomes efficient both due to ease of computation, and because in most cases, the state estimate at the previous time step is intuitively the most reasonable guess to start estimating state at time t [12]. The steps of the particle filter operation can be summarized as follows:

1. **Sample from the prior distribution** - First, a particle set $\{\tilde{x}_i\}, i = 1, \ldots, N$ is sampled from the prior state distribution $\pi_x(x_1)$.

2. **Assign importance weights** - Next step is to assign w_i to particles, based on a proposal distribution. The weight function is often chosen to be $w(x_t) = g(z_t|x_t)$, the measurement equation.

3. **Resampling** - The weighted samples drawn from the proposal distribution do not directly approximate a distribution. In order to obtain a probability mass

function that can be used as an approximate posterior at time t, a resampling procedure is required. Here, N new particles are chosen from the set $\{\tilde{x}_i^t, w_i^t\}, i = 1, 2, ..., N$, with probabilities proportional to the weights of each particle, and uniform weights of $\frac{1}{N}$ are reassigned to each resampled particle. Namely, this process ensures each particle to survive in proportion to its weight, and enables the resulting particle set $\{\bar{x}_i^t, \frac{1}{N}\}, i = 1, 2, ..., N$ to be a direct approximation of the posterior at time t, $p(x_t|z_{1:t})$.

4. **Time Update (Propagation)** - The final step of the process is to obtain the new proposal distribution. For all $t > 1$, this step requires the state dynamics to be taken into consideration. For this purpose, the equally weighted particle set obtained at time t is evolved according to the state dynamics given by A.3 to yield the new proposal distribution at time $t+1$. The process continues with step 2.

Following this sequential process, at each time step t, a particle set is obtained that can approximate the posterior state distribution at time t as follows

$$p(x_t|z_{1:t}) \approx \hat{p}(x) = \sum_{i=1}^{N} w_i^t \delta(x_t - \tilde{x}_i^t)\,. \tag{A.17}$$

Appendix B

Approximate Algorithms for Particle-Based Mutual-Information Computations

Mutual information between target state and robot measurements can be used to derive a control rule to maximize the expected future mutual information, i.e., to minimize the expected future uncertainty in the target location estimate. The observers in this case are mobile robots with navigation, that expect velocity inputs from a centralized controller. The proposed objective function is given by

$$\mathbf{u}^* = \arg\max_{\mathbf{u}_\tau} I(z_\tau; x_\tau) \tag{B.1}$$

$$s.t.\ ||u_t^{(i)}|| \leq V_{max},\ i = 1, 2, ..., M \tag{B.2}$$

$$I(\mathbf{z}_\tau; x_\tau) := H(\mathbf{z}_\tau) - H(\mathbf{z}_\tau | x_\tau)\,, \tag{B.3}$$

where $I(\cdot;\cdot)$ is the mutual information (MI) defined between two random variables that are its arguments, $H(\mathbf{z}_\tau)$ is the entropy of the measurement set $\mathbf{z}_\tau$, $H(\mathbf{z}_\tau | x_\tau)$ is the conditional entropy of the measurements given the state belief, $\tau = [t+1, ..., t+T]$, where T is the time horizon of the control problem, and $\mathbf{u}$ is the array of control inputs to the mobile sensors. We will focus on the T=1 case for this case study. Note that (B.3) is not trivial to solve, since it involves integrations over the entire measurement and state space, which can exponentially grow in size. As a first step, the particle filter representation will be used to simplify the problem into feasible approximations. By (4.15), the entropy terms given by (B.3) can be formulated as

$$\begin{aligned} H(\mathbf{z}_t) &= -\int_Z p(\mathbf{z}_t) \log p(\mathbf{z}_t) dZ \\ &\approx -\int_Z \left[\sum_{i=1}^N w_t^i p(\mathbf{z_t} | \tilde{x}_t^i) \log(\sum_{i=1}^N w_t^i p(\mathbf{z_t} | \tilde{x}_t^i)) \right] dZ \end{aligned} \tag{B.4}$$

Mutual Information Algorithm	Time Complexity
MI Based on a Particle Set†	$\mathcal{O}(\lambda^{-MD}M^2N)^*$
Single-Node Approximation† [61]	$\mathcal{O}(\lambda^{-D}M^2N)^*$
Pairwise-Node Approximation† [61]	$\mathcal{O}(\lambda^{-2D}M^2N)^*$
GMM Approximation on M robots [32]	$\mathcal{O}(MN^2)$
GMM Approximation on a Time Horizon [32]	$\mathcal{O}(MN^2T)$

† Requires Numerical Integration
* Assuming a fixed step-size numerical integrator with step size λ
D: Dimension of measurement space
M: Number of robots
N: Number of particles
T: Length of time horizon

Table B.1: Computational Complexity Associated with Information Based Control Methods

$$\begin{aligned} H(\mathbf{z}_t|x_t) &= -\int_Z p(\mathbf{z}_t|x_t)\log p(\mathbf{z}_t|x_t)dZ \\ &\approx -\int_Z \left[\sum_{i=1}^{N} w_t^i p(\mathbf{z_t}|\tilde{x}_t^i))\log p(\mathbf{z_t}|\tilde{x}_t^i))\right] dZ \end{aligned} \tag{B.5}$$

Note that this formulation requires a particle set $\{w_t^i, \tilde{x}_t^i\}, \quad i = 1, ..., N$ and a numerical integration tool for computing the entropy integral over the entire measurement space, where the dimension grows linearly in the number of observers. The exact mutual information metric given by (B.3) is not scalable for large teams of robots, an issue addressed by previous work [32, 61], which offers several approximations to the problem.

The first family of approximations of the MI quantity relies of simplifying the integrations over the entire measurement space by optimizing for only a single robot node or a pair of nodes at a time. The exact MI cost can be approximated at a single-node, which eliminates many dimensions of integration in approximating the quantity. This method presented in [61] can be used on board each robot node to yield a local optimum control input, and a global consensus is not required. The pairwise-node approximation [61] can be used to only consider the pairwise interactions between the robots, keeping all other state variables constant during optimization of control inputs for a pair of robots.

The second set of approximation models focus on the form of the problem instead, focusing on approximating the conditional measurement density as a Gaussian mixture, for which the integral is greatly simplified [32].

We will discuss the Gaussian Mixture Approximation in the following section, and present the computational complexities of other methods that have been introduced in the literature.

B.1 Gaussian-Mixture Approximation

A Gaussian Mixture approximation [32] uses a zeroth-order Taylor series expansion for approximating the differential entropy, $H(\mathbf{z}_t)$. The Taylor series approximation relies on a Gaussian Mixture Model approximation of the density $p(\mathbf{z}_t|\tilde{x}_t)$ and provides an approximation to differential entropy over the entire measurement space at a low computational cost (see Table B.1). The approximate Mutual Information is given by

$$\hat{I}_{GMM}(\mathbf{z}_t, x_t) = \hat{I}(z_t, x_{t+1}) = H_{GMM}(\mathbf{z}_t) - H_{GMM}(\mathbf{z}_t|x_t) \tag{B.6}$$

where

$$H_{GMM}(z_t) \approx -\sum_{n=1}^{N} w_n \sum_{m=1}^{N} w_m \mathcal{N}(\mu_n; \mu_m, \Sigma_m) \tag{B.7}$$

and

$$\begin{aligned} H_{GMM}(z_t|x_t) &\approx -\sum_{n=1}^{N} w_n \tilde{H}(z_t^{(j)}|x_t = \tilde{x}_{t,n}^{(j)}) \\ &= \sum_{n=1}^{N} w_n \left(\log((2\pi e)^M |\Sigma_n|)\right) \end{aligned} \tag{B.8}$$

where $\mu_i = \tilde{x}_t^i$ and M is the number of robots. The formulation of (B.7) and (B.8) follow from (B.4) and (B.5) respectively, by a zeroth order approximation to differential entropy for Gaussian Mixture Models (GMM), as developed by Huber et al. in [63] and applied to the information based control problem in [32]. Note that a higher order analytic approximation can similarly be used for obtaining tighter bounds.