Data Types and Behavioral Types

Yuhong Xiong
Edward A. Lee

Department of Electrical Engineering and Computer Sciences
University of California at Berkeley
Component Interfacing

- Two levels of interface:
  - data types and
  - dynamic interaction: communication & execution

- Dynamic behavior defined by models of computation (MoC)
Capturing Dynamic Behavior

- Dynamic behavior in each MoC can be described formally, using type systems.
- Success of type systems:
  - Safety through type checking
  - Polymorphism supports reuse (flexible components)
  - Type conversion
  - Program optimization
  - Interface documentation, clarification
  - Run-time reflection of component interfaces
Data Types in Ptolemy II

- Lattice-based infrastructure
- Support polymorphism, type conversion, and structured types
Behavioral Type

• Data types only specify static aspects of interface

• Proposal:
  - Capture the dynamic interaction of components in types
  - Obtain benefits analogous to data typing
  - Call the result *behavioral types*
Interaction Semantics

- **Flow of control issues**
  - in Ptolemy II, these are defined by a *Director* class
- **Communication between components**
  - in Ptolemy II, this is defined by a *Receiver* class

**Diagram:**
- Director
- IOPort
- producer actor
- consumer actor
- Receiver

Actor interface for execution: fire
Receiver interface for communication: put, get, hasToken
Receiver Object Model

IOPort

NoRoomException

«Interface»
Receiver

+get() : Token
+getContainer() : IOPort
+hasRoom() : boolean
+hasToken() : boolean
+put(t : Token)
+setContainer(port : IOPort)

NoTokenException

QueueReceiver

Mailbox

ProcessReceiver

PNReceiver

DEReceiver

SDFReceiver

FIFOQueue

ArrayFIFOQueue

CTReceiver

CSPReceiver

System, Inc.
Models of Computation

• Define the interaction semantics
• Implemented in Ptolemy II by a domain
  - Receiver + Director
• Examples:
  - Communicating Sequential Processes (CSP): rendezvous-style communication
  - Process Networks (PN): asynchronous communication
  - Synchronous Data Flow (SDF): stream-based communication, statically scheduled
  - Discrete Event (DE): event-based communication
  - Synchronous/Reactive (SR): synchronous, fixed point semantics
Formal Interaction Semantics: Use Interface Automata

- Based on interface automata
  - Proposed by de Alfaro and Henzinger
  - Concise composition (vs. standard automata)
  - *Alternating simulation* provides contravariance

- Compatibility checking
  - Done by automata composition
  - Captures the notion “components can work together”

- *Alternating simulation* (from Q to P)
  - All input steps of P can be simulated by Q, and
  - All output steps of Q can be simulated by P.
  - Provides the ordering we need for subtyping & polymorphism

- Key theorem about compatibility and alternating simulation
**Example: Synchronous Dataflow (SDF)**

**Consumer Actor Type Definition**

Such actors are passive, and assume that input is available when they fire.

**Inputs:**

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>fire</td>
</tr>
<tr>
<td>t</td>
<td>Token</td>
</tr>
<tr>
<td>hTT</td>
<td>Return True from hasToken</td>
</tr>
<tr>
<td>hTF</td>
<td>Return False from hasToken</td>
</tr>
</tbody>
</table>

**Outputs:**

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fR</td>
<td>Return from fire</td>
</tr>
<tr>
<td>g</td>
<td>get</td>
</tr>
<tr>
<td>hT</td>
<td>hasToken</td>
</tr>
</tbody>
</table>
Type Definition – Synchronous Dataflow (SDF) Domain

[Diagram showing the relationships between different components such as producer, consumer, receiver, director, IOPort, and interface.]
Type Checking - Compose SDF Consumer Actor with SDF Domain
Type Definition - SDF
Consumer Actor in SDF Domain

1. receives token from producer
2. accept token
3. internal action: fire consumer
4. internal action: call get()
5. internal action: get token
6. internal action: return from fire

interface to producer actor
Type Definition – Discrete Event (DE) Domain

This domain may fire actors without first providing inputs.
Recall Component Behavior
SDF Consumer Actor

- is fired
- calls get()
- gets a token
- returns
Type Checking - Compose SDF Consumer Actor with DE Domain

- Empty automaton indicates incompatibility
- Composition type has no behaviors
Subtyping Relation
Alternating Simulation: SDF \leq DE

SDF Domain

DE Domain
System-Level Type Lattice - Defined by Alternating Simulation

- Subtyping relation
- Shown here for a few Ptolemy II domains

If an actor is compatible with a certain type, it is also compatible with the subtypes
Polymorphic Consumer Actor

1. is fired
2. calls hasToken()
3. true
4. call get()
5. get token
6. return

This actor checks for token availability before attempting to get the token.
Domain Polymorphic Actor Composes with the DE Domain
Domain Polymorphic Actor Also Composes with the SDF Domain
Conclusion

• Data types
  - Lattice-based infrastructure
  - Support polymorphism, type conversion, and structured types

• Behavioral types
  - Formally captures the structure of MoCs
  - Describe interaction types and component behavior using interface automata
  - Perform type checking through automata composition
  - Subtyping order is given by the alternating simulation relation, supporting polymorphism