dSpace
Composable Abstractions for Smart Spaces

Silvery Fu,
Sylvia Ratnasamy
3x IoT devices, 2015-2020

Still complex to program smart spaces!

13.6 per person by 2023
Programming Smart Spaces

Today: why it's complex?

device-centric APIs
monolithic implementations
if-then-that policies
"The basic technique we have for managing the complexity of software is modularity" - Barbara Liskov
"The basic technique we have for managing the complexity of software is modularity" - Barbara Liskov

For smart spaces:

What is the right modularity?

What is the **minimal** set of abstractions to achieve it?
A lamp / smart light bulb
dSpace:

Modules

Digivice
Each digivice $D$ has:

Model - $D.m$od: attribute-values that capture $D$'s intended states (intent) current states (status), and events

Driver - $D.drv()$: code that reconcile status to intent
Heterogeneity → Complexity

Idea: use a universal digivice to configure a device-specific digivice

Digivices can have different device libraries (driver) programming lang. (driver) schema (model)

**Driver**

D.drv()

**Model**

D.mod

**Digivice L1**

L1.mod.power
   .intent: on
   .status: off

**Digivice L2**

L2.mod.turn_on
   .intent: true
   .status: false
*Model*

```java
D.mod.power
  .intent: on
  .status: off
```

*Driver*

```java
D.drv():
  switch vendor
  case L1: ...
  case L2: ...
```

**Universal Lamps**

**Heterogeneity → Complexity**

**Idea:** use a universal digivice to configure a device-specific digivice
dSpace: Modules

Digivice

Composition
Mount

Model
D.mod.power
.intent: on
.status: off

Driver
D.drv():
switch vendor
case L1: ...
case L2: ...

Compose Digivices with Mount primitive

Universal Lamps

Heterogeneity $\rightarrow$ Complexity

Idea: use a universal digivice to configure a device-specific digivice
Compose Digivices with Mount primitive

Mount primitive:

**Mount(A, B)** allows `B.drv()` to:

1. Write to `A.mod.intent`
2. Read from `A.mod.status`

**B**: parent; **A**: child

**Universal Lamps**

- Parent
- Child

Status/Events

- intent

Mount (primitive)
Living Room

Aggregate brightness of the living room

---

**dSpace:**
- Modules
- Digivice
- Composition
- Mount

**Parent**
- status
- events
- intent

**Child**
Idea: Introducing a living room digivice and mount both lamps!
Idea: Introducing a living room digivice and mount both lamps!

Developers of the room:
Don't interact with physical devices
Program universal lamps
Raising the level of abstractions

House

Living Room

Kitchen

Garden

dSpace:

Modules

Digivice

Composition

Mount

Digivices form control hierarchy
dSpace:

Modules

Digivice

Composition

Mount

Living Room

Mount
**Goal:** integrate data processing with digivices

Video processing

ML/object recognition

Living Room

![Diagram of Living Room setup with FFmpeg and OpenCV logos]
dSpace: Modules

Digivice

Digidata

Composition

Mount

**Goal:** integrate data processing with digivices

Video processing

ML/object recognition

**Digidata**

**Model**

CV.mod

.in: rtsp://..

.out: human

**Driver**

CV.drv():

frame = capture(in)

.out = detect(frame)

**Living Room**

Each **digidata T** has:

T.mod.in: data input

T.mod.out: data output

Driver **T.drv():** data processing code to transform T.mod.in to T.mod.out
Living Room

**Model**
- CV.mod
  - .in: rtsp://..
  - .out: human

**Driver**
- CV.drv():
  - frame = capture(.in)
  - .out = detect(frame)

---

**dSpace:**
- Modules
- Digivice
- Digidata

**Composition**
- Mount
Mounting digidata T to digivice D allows:
D to write T.mod.in
D to read T.mod.out

Living Room

Digidata
Model
CV.mod
.in: rtsp://..
.out: human

Driver
CV.drv():
frame = capture(.in)
.out = detect(frame)
Mounting digidata T to digivice D allows:
D to write T.mod.in
D to read T.mod.out

Living Room

Process the video before sending it to CV?

Cam
Roomba
CV
Cam

dSpace:
Modules
Digivice
Digidata
Composition
Mount

Digidata
Model
CV.mod
.in: rtsp://..
.out: human

Driver
CV.drv():
frame = capture(.in)
.out = detect(frame)
Pipe(A, B) writes A.mod.out to B.mod.in

**Model**
CV.mod
.in: rtsp://..
.out: human

**Driver**
CV.drv():
frame = capture(.in)
.out = detect(frame)

Living Room

Write FF.mod.out.url to CV.mod.in.url
Device moves across rooms
Device moves across rooms

Allow multiple parents, but only one can have write-access and the other(s) are yielded.
dSpace:

- Modules
- Digivice
- Digidata

Composition

- Mount
- Pipe
- Yield

Policies

- Delegation

Device moves across rooms
Delegate access to third-party

Allow multiple parents, but only one can have write-access and the other(s) are yielded

Must preserve multi-tree topology to avoid loops
Device moves across rooms
Delegate access to third-party
Handle intent conflicts due to physical events

Allow multiple parents, but only one can have write-access and the other(s) are yielded

Must preserve multi-tree topology to avoid loops

More details about policies in the paper!
Today:
- Device-centric abstractions
- Ad-hoc composition: If-then-that
- Limited HL abstraction and policies
- Monolithic architecture

dSpace:
- Composable digivece/data
- First-class primitives: Mount Pipe Yield
- Rich policies: delegation intent reconc.

Recap: Programming Smart Spaces
What makes dSpace simple?
Today:

- Device-centric abstractions
- Monolithic architecture
- Ad-hoc composition: If-then-that
- Limited HL abstraction and policies
- Monolithic architecture

dSpace:

- Composable digivice/data
- First-class primitives: Mount Pipe Yield
- Rich policies: delegation intent reconc.
- Digis run as microservices

Recap: Programming Smart Spaces

What makes dSpace simple?
Implement dSpace with Microservices

kubernetes

$kubectl

{model}  {model}  {model}

kube-apiserver

kube-scheduler

etcd

pod
digi
driver

pod
digi
driver

pod
digi
driver
Implement dSpace with Microservices

kubernetes

dv

digi driver

digi driver

digi driver

{kmodel} {model} {model}

kube-apiserver

dSpace controllers

dSpace controllers

kubelet

kube-scheduler

etcd
See the paper for details:

Design and Implementation
Design Principles, Driver Programming, Runtime Arch., Security etc.

Evaluation
10 scenarios in smart home with 9 devices
< 300 lines of code (LoC; +15%) for all scenarios
vs. existing frameworks
4/10 scenarios are dSpace only, the rest more (4x) LoC
User study and performance benchmarks
4.41 MOS (0-5); runtime adds <20% latency overhead
Goal: simplify development of smart space apps

Manage complexity through the (right) modularity:

Digivice, Digidata + Mount, Pipe, Yield

Thank you!

github.com/NetSys/dspace