afspm Overview

Nick Sullivan

National Research Council Canada (NRC/CNRC)

September 14, 2023

Outline

SPM Basics

In Scanning Probe Microscopy (SPM), an **atomically-sharp tip** is **scanned** above a **surface**, while **measuring** one or more **properties** gleaned from this tip.

This process allows **atomic-level imaging** of properties, spectroscopic analysis, and even manipulation of a sample (toward **atomic-scale manufacturing**).

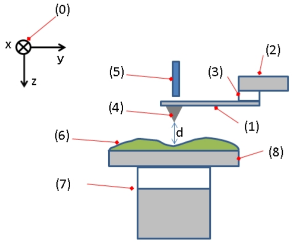


Figure: Typical AFM Configuration, Tom Toyosaki, Wikimedia Commons.

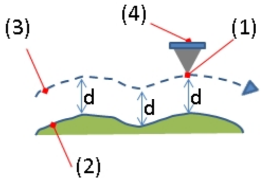
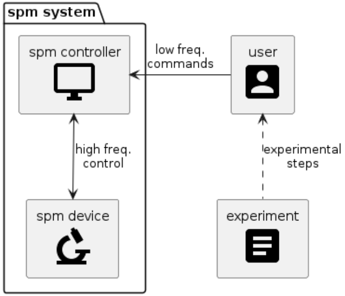


Figure: Schematics of AFM topographic image forming, Tom Toyosaki, Wikimedia Commons.

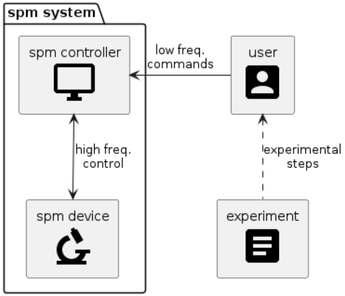
Standard Experiment



In a traditional SPM experiment, a researcher with domain knowledge will:

1. **Prepare the system**: including defining the SPM mode (e.g. FM-AFM).
2. **Run the experiment**: monitoring collected scans, deciding on next scans, and updating any aspects of the experiment.
3. **Finalize the experiment**: by undoing any experiment-specific setup needed to run.

Standard Experiment



In a traditional SPM experiment, a researcher with domain knowledge will:

1. **Prepare the system**: including defining the SPM mode (e.g. FM-AFM).
2. **Run the experiment**: monitoring collected scans, deciding on next scans, and updating any aspects of the experiment.
3. **Finalize the experiment**: by undoing any experiment-specific setup needed to run.

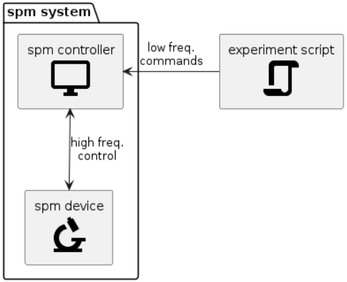
Running the experiment is often ***long****, and requires* ***constant*** *researcher* ***attention****.*

System Scripting and Code Reuse

Many SPM systems allow custom scripts to run an experiment.

However:

- Scripts written for a **specific** SPM system **cannot be re-used** for other SPM systems: different API/language constraints.
- While **decoupling** of SPM device and experiment logic **is possible**, it is **rarely** a **priority** for researchers.

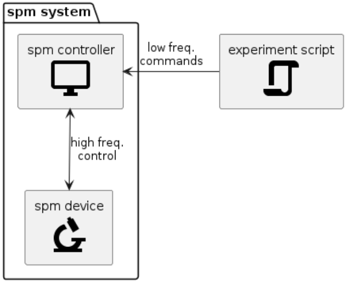


System Scripting and Code Reuse

Many SPM systems allow custom scripts to run an experiment.

However:

- Scripts written for a **specific** SPM system **cannot be re-used** for other SPM systems: different API/language constraints.
- While **decoupling** of SPM device and experiment logic **is possible**, it is **rarely** a **priority** for researchers.



*Code **reuse** is **rare**.*

Goals and Scope

Goals and Scope

Goals

- **Clear Decoupling**: of SPM device specifics from the desired experiment; to allow reuse of a given experiment on multiple devices.
- **Multi-Language Support**: for integration with SPM devices; we should not be the limiting factor to integration.
- **Pausable Automation**: to allow a researcher to take over.
- **Separable on Multiple Devices**: composed of concise, separable components.

Goals and Scope

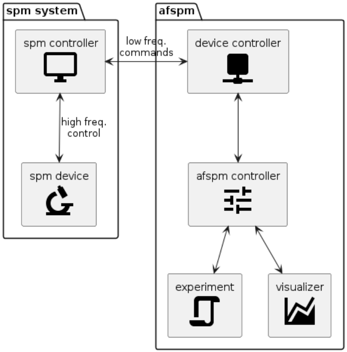
Goals

- **Clear Decoupling**: of SPM device specifics from the desired experiment; to allow reuse of a given experiment on multiple devices.
- **Multi-Language Support**: for integration with SPM devices; we should not be the limiting factor to integration.
- **Pausable Automation**: to allow a researcher to take over.
- **Separable on Multiple Devices**: composed of concise, separable components.

Scope

afspm will concern itself **only** with automation of high-level, low-frequency decisions a researcher would perform **during** an experiment.

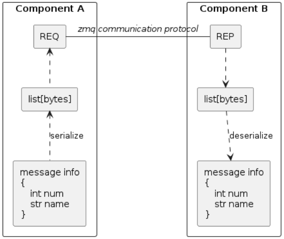
High-Level Design



afspm is designed around 'computation' components that correspond to nodes in a network. An experiment contains:

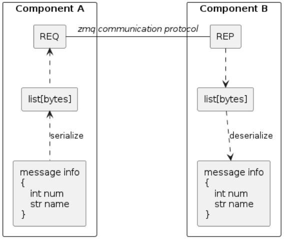
- **Device Controller**: abstract - specifies the required methods to communicate with an SPM device. Each SPM device will have its own DeviceController implementation.
- **Afspm Controller**: mediates control of the SPM device (only 1 component in control at a time).
- **Afspm Components**: the one or more components the user requires to run their experiment.

Communication Protocol


Component A
Component B
REQ
REP
zmq communication protocol
list[bytes]
list[bytes]
serialize
deserialize
message info { int num str name }
message info { int num str name }

afspm uses ZeroMQ as its communications/concurrency framework and protobuffers as its serialization layer. Both are cross-platform and cross-language.

Communication Protocol

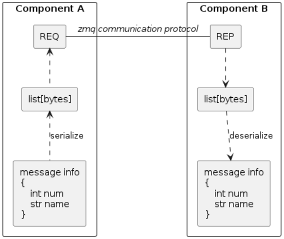

Component A
Component B
REQ
REP
zmq communication protocol
list[bytes]
list[bytes]
serialize
deserialize
message info { int num str name }
message info { int num str name }

afspm uses ZeroMQ as its communications/concurrency framework and protobuffers as its serialization layer. Both are cross-platform and cross-language.

What is ZeroMQ?

Library that abstracts away protocols used, allowing easy switching of how 'nodes' communicate in a system.

Communication Protocol


Component A
Component B
REQ
REP
zmq communication protocol
list[bytes]
list[bytes]
serialize
deserialize
message info { int num str name }
message info { int num str name }

afspm uses ZeroMQ as its communications/concurrency framework and protobuffers as its serialization layer. Both are cross-platform and cross-language.

What is ZeroMQ?

Library that abstracts away protocols used, allowing easy switching of how 'nodes' communicate in a system.

What are protobuffers?

Library for serializing structured data (translating a data structure into a format that can be stored/communicated).

I/O Paths

I/O Paths

Publisher-Subscriber Path

- The DeviceController **publishes** ScanState, ScanParameters, and Scan **changes**.
- These are passed on by the AfspmController. Data is **stored** in a **cache** and resent to new/crashed components.
- Each component choose what aspects to **subscribe** to, and receives data from these.

I/O Paths

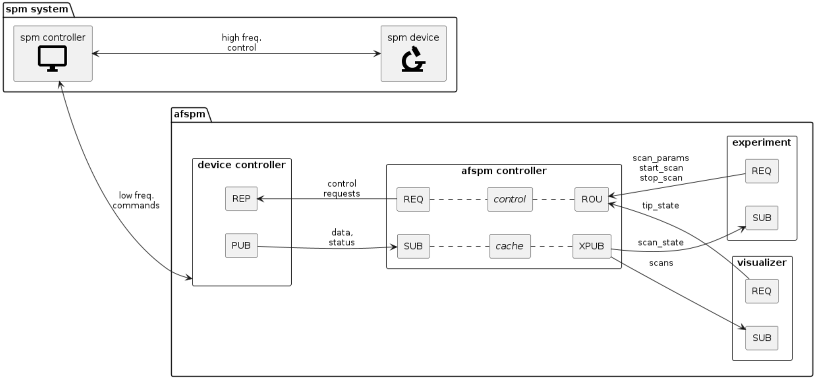
Publisher-Subscriber Path

- The DeviceController **publishes** ScanState, ScanParameters, and Scan **changes**.
- These are passed on by the AfspmController. Data is **stored** in a **cache** and resent to new/crashed components.
- Each component choose what aspects to **subscribe** to, and receives data from these.

Control Path

- Each component can send **control requests** over its client.
- The AfspmController determines which **client** is **in-control**, and **forwards** these to the DeviceController.
- The DeviceController **receives** control requests from one client and **responds**.

afspm: Detailed View



afspm Controller

afspm Controller

Cache Logic

Data is **stored** into the **cache** according to a **user-defined configuration**.

These map a **protobuf message** to a **cache key** (envelope), and vice-versa.

afspm Controller

Cache Logic

Data is **stored** into the **cache** according to a **user-defined configuration**.

These map a **protobuf message** to a **cache key** (envelope), and vice-versa.

Experiment Problems

Any component can **report** experiment **problems**, indicating issues that should cause the experiment to **pause** until **resolved**, and can **remove** these problems.

This allows, e.g., detecting a tip crash and attempting to correct it.

afspm Controller

Cache Logic

Data is **stored** into the **cache** according to a **user-defined configuration**.

These map a **protobuf message** to a **cache key** (envelope), and vice-versa.

Experiment Problems

Any component can **report** experiment **problems**, indicating issues that should cause the experiment to **pause** until **resolved**, and can **remove** these problems.

This allows, e.g., detecting a tip crash and attempting to correct it.

Control Modes

The AfspmController defines the **control mode**, which can be:

- **Automated**: default, automation runs.
- **Manual**: pause automation.
- **Problem**: experiment problems are logged, pause automation.

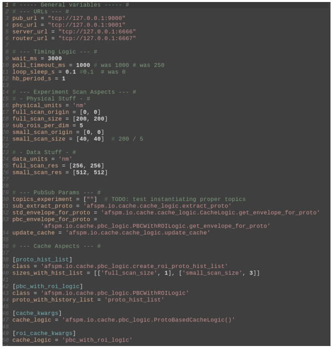
The Config File

afspm uses a **single** TOML **configuration file** per experiment.

Within this file, a user defines:

- The communication protocols used between components.
- Common variables passed between components (e.g. how big the scan size will be).
- The components to spawn.

Top-level definitions can function as **variables**: any **references** deeper in the config are **replaced** by them. This should minimize repeating oneself.



```toml
# ----- General variables ----- #
# --- URLs --- #
pub_url = "tcp://127.0.0.1:9000"
psc_url = "tcp://127.0.0.1:9001"
server_url = "tcp://127.0.0.1:6666"
router_url = "tcp://127.0.0.1:6667"

# --- Timing Logic --- #
wait_ms = 3000
poll_timeout_ms = 1000 # was 1000 # was 250
loop_sleep_s = 0.1 #0.1  # was 0
hb_period_s = 1

# --- Experiment Scan Aspects --- #
# - Physical Stuff - #
physical_units = 'nm'
full_scan_origin = [0, 0]
full_scan_size = [200, 200]
sub_rois_per_dim = 5
small_scan_origin = [0, 0]
small_scan_size = [40, 40]  # 200 / 5

# - Data Stuff - #
data_units = 'nm'
full_scan_res = [256, 256]
small_scan_res = [512, 512]


# --- PubSub Params --- #
topics_experiment = [""]  # TODO: test instantiating proper topics
sub_extract_proto = 'afspm.io.cache.cache_logic.extract_proto'
std_envelope_for_proto = 'afspm.io.cache.cache_logic.CacheLogic.get_envelope_for_proto'
pbc_envelope_for_proto =
        'afspm.io.cache.pbc_logic.PBCWithROILogic.get_envelope_for_proto'
update_cache = 'afspm.io.cache.cache_logic.update_cache'

# --- Cache Aspects --- #

[proto_hist_list]
class = 'afspm.io.cache.pbc_logic.create_roi_proto_hist_list'
sizes_with_hist_list = [['full_scan_size', 1], ['small_scan_size', 3]]

[pbc_with_roi_logic]
class = 'afspm.io.cache.pbc_logic.PBCWithROILogic'
proto_with_history_list = 'proto_hist_list'

[cache_kwargs]
cache_logic = 'afspm.io.cache.pbc_logic.ProtoBasedCacheLogic()'

[roi_cache_kwargs]
cache_logic = 'pbc_with_roi_logic'
```

Spawning the Experiment

Spawning the Experiment

Distributed Computing

Components can be **split up** among devices; on startup, the components to spawn can be specified.

Spawning the Experiment

Distributed Computing

Components can be **split up** among devices; on startup, the components to spawn can be specified.

Component Monitoring

All spawned components are **monitored**:

- Each sends **heartbeats** at a regular cadence.
- If one **stops** beating, it is **restarted**.

This should minimize a crash breaking experiments.

The End

Let us know what you think and help us make it better.

afspm on github