Decentralized Runtime Verification

Building Blocks: Components, Observations, Specification and Monitors [1, 4]

A decentralized system contains multiple components. Components behavior is abstracted as observations.

The observation <alarm, true> indicates that the alarm is triggered.

G(!alarm)

A specification is a user-provided formal description of the correct behavior of the system. It is used to synthesize and integrate monitors into the system.

Specifications can be defined using automata, Linear-time Temporal Logic (LTL), or other formalisms.

Monitors are responsible for checking whether the current execution of the system complies with the specification. One or more monitors are attached to components. Monitors receive observations, do some processing and communicate with other monitors

Monitoring API

An API for Common Monitoring Activities

Parsing and managing specifications and traces. Datastructures for storing observations and monitor state.

Creating, accessing and instrumenting measures into the execution.

Deploying and setting up components, monitors, and associating monitors to specifications.

Specifying high level API for monitoring and communication between monitors.

Simple and Extensible Formats

THEMIS uses XML for specifications.
Specifications are passed to your algorithm.
Your algorithm is responsible of parsing and setting up the monitors appropriately.

```
<specifications>
    <specification id="monitor" class="uga.corse.themis.monitoring.SpecLTL">
        <setLTL><![CDATA[[]((m<->X(l)))]]></setLTL>
    </specification>
    <specification id="r1" class="uga.corse.themis.lights.specs.SpecRoom">
        <addDevice>a</addDevice>
        <addDevice>b</addDevice>
    </specification>
    <specification id="r2" class="uga.corse.themis.lights.specs.SpecRoom">
        <addDevice>c</addDevice>
        <addDevice>d</addDevice>
    </specification>
    <!-- Connect: = : both ways, l-r : one way l -> r -->
    <specification id="network"
            class="uga.corse.themis.lights.specs.SpecConnectivity">
        <connect>r1=r2</connect>
        <connect>r1=r3</connect>
    </specification>
</specifications>
```

1. Design

Decentralized RV Algorithms

Design new algorithms
Variants of existing algorithms
Refinements of existing algorithms

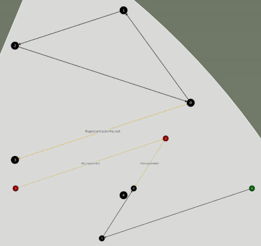
Measures

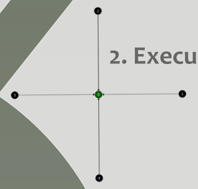
Create measures by instrumentation (AspectJ)
Use existing API for measures
Measures target all algorithms using the API

```
void setupRun(MonitoringAlgorithm alg) {
    addMeasure(
        new Measure("msg_num","Msgs",0L,Measures.addLong));
}
after(Integer to, Message m) : Commons.sendMessage(to, m) {
    update("msg_num" , 1L);
}

public void monitor(int t, Memory<Atom> observations)
throws ReportVerdict, ExceptionStopMonitoring {
    mem.merge(observations);
    if(receive()) isMonitoring = true;
    if(isMonitoring) {
        if(!observations.isEmpty())
            ehe.tick();
        boolean b = ehe.update(mem, -1);
        if(b) {
            VerdictTimed v = ehe.scanVerdict();
            if(v.isFinal())
                throw new ReportVerdict(v.getVerdict(), t);
            ehe.dropResolved();
        }
        int next  = getNext();
        if(next != getID()) {
            Representation toSend = ehe.sliceLive();
            send(next, new RepresentationPacket(toSend));
            isMonitoring = false;
        }
    }}
```

```
Map<Integer, ? extends Monitor> setup() {
    config.getSpec().put("root",
        Convert.makeAutomataSpec(
        config.getSpec().get("root")));
    Map<Integer, Monitor> mons = new HashMap<>();
    Integer i = 0;
    for(Component comp : config.getComponents()) {
        MonMigrate mon = new MonMigrate(i);
        attachMonitor(comp, mon);
        mons.put(i, mon);
        i++;
    }
    return mons;
}
```





2. Execute

Simulation

Monitor a trace using an algorithm

Visualization

Basic topology and communication visualization

Experiment

An experiment is a reproducible set of parameters, specifications, and algorithms

Experiment Configuration

```
tests.skip=false
tests.discard=true
tests.oracle=uga.corse.themis.tools.experiment.TestLTL
traces.dir=../traces
traces.generate=false
traces.components=5
traces.observations=2
traces.maximum=200
traces.length=100
run.length=110
run.components=3
run.specs=specs.txt
run.algorithms=../algs.txt
THEMIS_BENCH_DB=run.db
THEMIS_CACHE_LTL=cache
```

THEMIS
A Tool for Decentralized Monitoring

Antoine El-Hokayem   Yliès Falcone

first.last @ univ-grenoble-alpes.fr
Univ. Grenoble Alpes, LIG, Inria, CNRS, F-38000 Grenoble, France

3. Analyze

Flexible

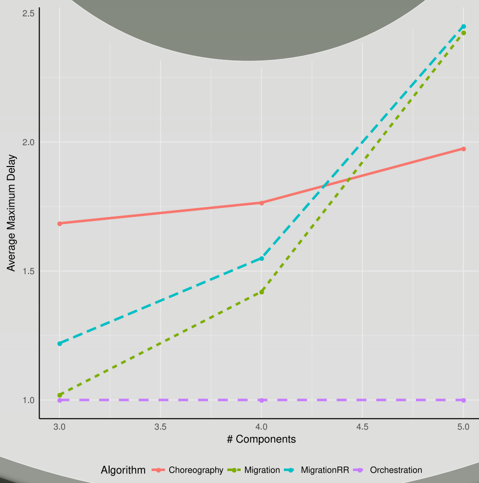
Measures are stored in a database
Use any third party tools for analysis

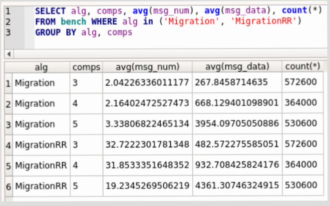
Modular

Instrumented at runtime using AspectJ
Existing API and classes to extend

Reusable

Measures apply to different algorithms
Experiments can re-use new measures





```
SELECT alg, comps, avg(msg_num), avg(msg_data), count(*)
FROM bench WHERE alg in ('Migration', 'MigrationRR')
GROUP BY alg, comps
```

	alg	comps	avg(msg_num)	avg(msg_data)	count(*)
1	Migration	3	2.04226336011177	267.8458714635	572600
2	Migration	4	2.16402472527473	668.129401098901	364000
3	Migration	5	3.33806822465134	3954.09705050886	530600
4	MigrationRR	3	32.7222301781348	482.572275585051	572600
5	MigrationRR	4	31.8533351648352	932.708425824176	364000
6	MigrationRR	5	19.2345269506219	4361.30746324915	530600

THEMIS

Designing, Analyzing, and Comparing Decentralized RV Algorithms

THEMIS is a tool to facilitate the design, development, and analysis of decentralized monitoring algorithms. It is developed using Java and AspectJ.

It consists of a library and command-line tools. THEMIS provides an API, data structures and measures for decentralized monitoring.

These building blocks can be reused or extended to modify existing algorithms, design new more intricate algorithms, and elaborate new approaches to assess existing algorithms.

The theoretical aspects can be found in [2].

Use Cases

Designing New Algorithms

THEMIS makes it easy to prototype and incrementally design new algorithms. Common tasks such as parsing automata and LTL, setting up monitors, and communication are managed by the framework.

Optimizing Existing Algorithms

Using the experiment tool and the existing measures, designing new variants of algorithms can be easily re-run in a reproducible environment. New measures can be added to enrich the comparison, which will also apply to the older versions.

Comparing Decentralized RV Algorithms

The monitoring API can be used to compare different algorithms [3]. This is done by analyzing the usage of the same datastructures or API calls (such as communication).

References

[1] Christian Colombo and Ylies Falcone. 2016. Organising LTL monitors over distributed systems with a global clock. Formal Methods in System Design 49, 1-2 (2016), 109–158.

[2] Antoine El-Hokayem and Yliès Falcone. 2017. Monitoring Decentralized Specifications. In 26th International Symposium on Software Testing and Analysis, ISSTA 2017

[3] Andreas Bauer and Yliès Falcone. 2016. Decentralised LTL monitoring. Formal Methods in System Design 48, 1-2 (2016), 46–93

[4] Martin Leucker and Christian Schallhart. 2009. A brief account of runtime verification. J. Log. Algebr. Program.78, 5 (2009), 293–303.

gitlab.inria.fr/monitoring/themis

Poster template by Ugo Sangiorgi based on Felix Breuer's work