

---

**The Scalable Simulation Framework  
and  
SSF network models**

---

Dirk Jacob  
March 25, 2001

# Overview

---

1. SSFnet Overview
2. Scalable Simulation Framework (SSF)
3. SSF Network Models (SSFnet)
4. Domain Modeling Language (DML)
5. More info
6. Putting it all together: Examples

# SSFnet at a glance

---

SSFnet project has two main components:

- research on and development of scalable modeling and simulation tools and
- using these tools - research on the dynamic behavior of very large networks

Software research:

- focus on scalability: modeling scalability, performance scalability

Network research:

- help address a number of modeling and simulation challenges

# How to build scalable simulations

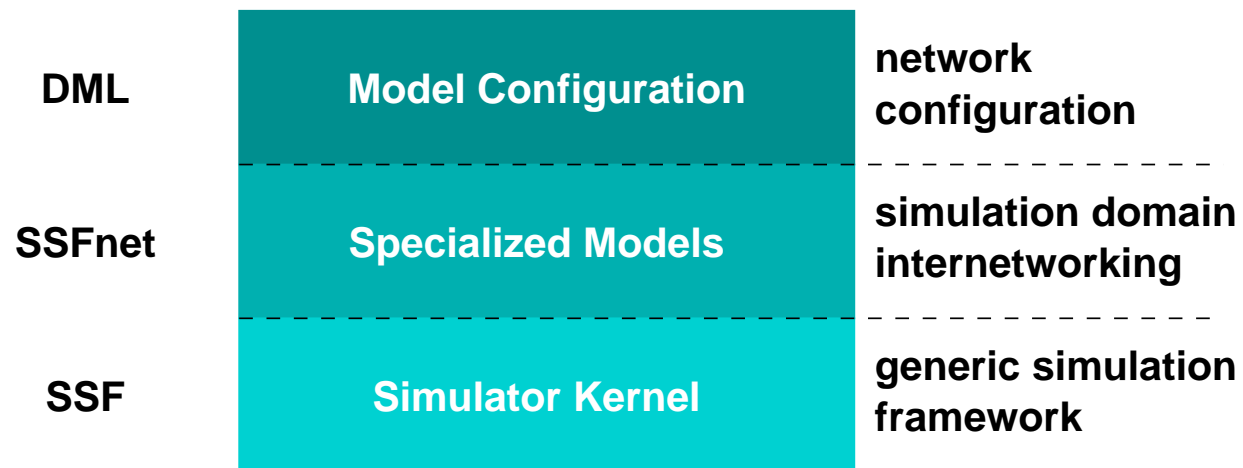
---

- generic simulation framework, which provides mechanisms for simulation of arbitrary applications
- specialized "models" provide a mapping from the simulated world to the simulator world
- a model imitates the behaviour of a real component by using the mechanisms provided by the simulator
- models may need specific configuration, when instantiated

# SSFnet building blocks

---

The SSFnet framework consists of several components:



# Scalable Simulation Framework (SSF)

---

Provides a single, unified API to the simulator kernel:

- five base class interfaces `Event`, `Entity`, `inChannel`, `outChannel` and `process`.
- Java and C++ bindings
- not limited to simulate communication networks
- simulation domain determined by specific models atop of SSF
- built-in multiprocessor support

# Entity

---

- base class for **all** simulation components
- **Examples:** hosts, routers, links, TCP sessions, etc.
- container mechanism for defining alignment relations among a model's pieces
- coaligned entities presumably interact through event exchange on channels
- underlying simulator takes this into account, when mapping entities to processors
- Methods: `now()`, `startAll()`, `pauseAll()`, `resumeAll()`, `joinAll()`, `alignTo()`, `alignment()`, `coalignedEntities()`, `init()`, `processes()`, `inChannels()` and `outChannels()`

# Event

---

- base class for the quantum of information flowing over channels
- **Examples:** protocol packets and timers
- Methods: `save()`, `release()` and `aliased()`

# process

---

- base class for describing an entity's behavior
- **Examples:** run-time behaviour of protocols, etc.
- each instance of a process is associated with a specific entity
- may wait for input arriving on inChannels or wait for time to elapse
- may also wait on channels of entities coaligned with its owner
- Methods: `owner()`, `action()`, `init()`, `waitOn()`, `waitForever()`, `waitFor()`, `waitOnFor()` and `isSimple()`

# inChannel, outChannel

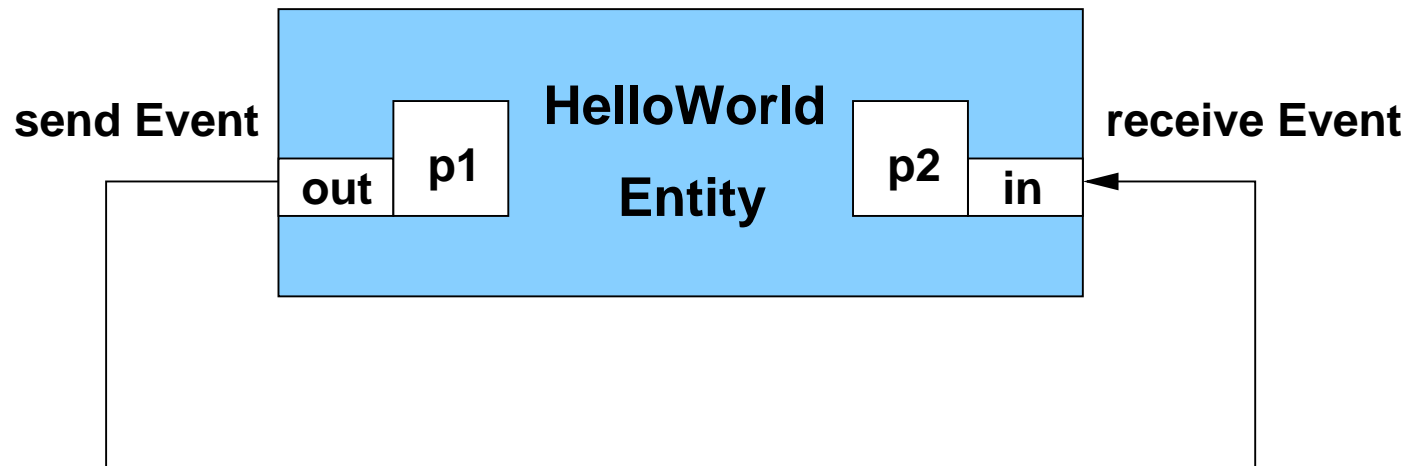
---

- communication endpoints for event exchange
- **Examples:** protocol interaction, etc.
- each instance belongs to a specific entity
- multicast in- (many to one) and outChannels (one to many) and bus-style channel mappings (many to many)
- outChannels have a transmission delay associated with them
- inChannel Methods: `owner()`, `activeEvents()` and `mappedTo()`
- outChannel Methods: `owner()`, `mappedTo()`, `write()`, `mapTo()` and `unmap()`

# Example: HelloWorld

---

This is a simple example to demonstrate the execution of a SSF simulation.



## Example: HelloWorld (cont.)

---

```
import com.renesys.raceway.SSF.*;

public class HelloWorld extends Entity {

    public long DELAY = 20;
    public int rcvd;

    public inChannel IN;
    public outChannel OUT;
```

## Example: HelloWorld (cont.)

---

```
public HelloWorld() {
    rcvd = 0;
    IN = new inChannel( this );
    OUT = new outChannel( this, DELAY );

    OUT.mapto( IN );

    new process(this) {
        public void action() {
            OUT.write(new Event());
            waitFor( DELAY );
        }
    };
};
```

## Example: HelloWorld (cont.)

---

```
new process(this) {
    public void action() {
        waitOn( IN );
        rcvd++;
    }
};
}
```

## Example: HelloWorld (cont.)

---

```
public static void main(String[] argv) {
    HelloWorld hello = new HelloWorld();

    hello.startAll( 200 );
    hello.joinAll();
    System.out.println("total received events = "+hello.rcvd);
}
}
```

# SSF implementations

---

SSF implementations available:

- **Raceway**: High-performance commercial implementation of the Java SSF API from Renesys Corp.
- **JSSF**: Reference implementation of the Java SSF API from Cooperating Systems Corp.
- **CSSF**: Reference implementation of the C++ SSF API from Cooperating Systems Corp.
- **DaSSF**: High-performance implementation of the C++ SSF API from Dartmouth College

# SSF Network Models (SSFnet)

---

”SSFnet provides open-source Java models of protocols (IP, TCP, UDP, BGP4, OSPF and others), network elements (hosts, routers, links, LANs) and assorted support classes for realistic multi-protocol, multi-domain internet modeling and simulation.”

- collection of components for simulation at IP level and above
- configurable at run-time through a configuration database
- two parts: SSF.OS (modeling of the host and operating system components, such as protocols), SSF.Net (network connectivity, node and link configuration)

# Available protocols and applications

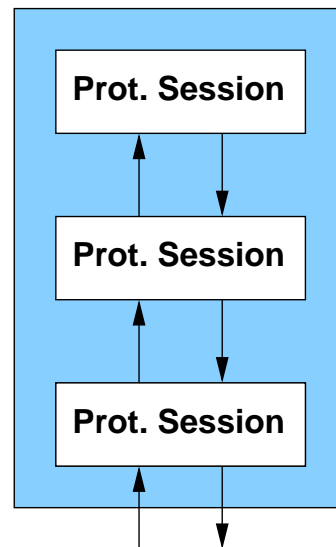
---

- Routing:
  - SSF.OS.IP
  - SSF.OS.OSPF
  - SSF.OS.BGP4
- Transport:
  - SSF.OS.TCP
  - SSF.OS.UDP
  - SSF.OS.Socket
- Applications/workload:
  - TCP and UDP clients/servers
  - HTTP clients/servers

# SSF.OS

---

Principal classes: ProtocolGraph, ProtocolSession, ProtocolMessage and PacketEvent



## SSF.OS (cont.)

---

- protocol implementations must be derived from `ProtocolSession`
- each protocol graph is an entity with in- and out-channels for exchanging `PacketEvents`
- protocol messages represent packet headers and payload of packets specific to a protocol
- `PacketEvent` is a wrapper class, that turns a protocol message into an event, which can be sent over a channel

# SSF.Net

---

- principal classes: `Net`, `Host`, `Router`, `NIC` and `link`
- `Net` loads the configuration and instantiates the entire model
- `Host` is derived from `ProtocolGraph` and must minimally include IP and at least one Network Interface Card (`NIC`)
- `Router` is a special class derived from `Host`
- `NIC` is a pseudo-protocol which maintains a pair of in- and out-channels connected to the world outside the `ProtocolGraph`
- a link connects a `NIC` to other `NICs` on the same link

# Monitoring possibilities

---

- support for export of flow data in Cisco NetFlow format
- support for dumps of packet data in tcpdump format
- TCP instrumentation for dumps of internal state variables of active TCP connections

# Domain Modeling Language (DML)

---

- DML configuration provides the configuration for specific instances of SSFnet objects
- used for configuring network models and modeling topologies
- schemas provide a mechanism for flexible syntax checking of models
- easy to introduce new schemas with new protocol models

# Example for a DML configuration

---

```
router[
  interface[id 0 bitrate 100000000 latency 0.0001]
  interface[id 1 bitrate 100000000 latency 0.0001]
  interface[id 2 bitrate 100000000 latency 0.0001]
  interface[id 3 bitrate 100000000 latency 0.0001]
  graph[
    ProtocolSession[name ip use SSF.OS.IP]
    ProtocolSession[name ospf use SSF.OS.OSPF.sOSPF]
  ]
]
```

# Example for a DML schema

---

```
host [  
    graph %T1!::.schemas.graph    # protocol graph specification  
  
    interface [_extends .schemas.interface  
        id %I idrange [from %I1! to %I1!]  
        ip %S  
    ]  
  
    _find .schemas.route          # Static routes for this host  
    _find .schemas.nhi_route  
]  
  
router [_extends .schemas.host]
```

# DML syntax

---

- list of whitespace-separated key and value pairs
- keys are alphanumeric strings
- reserved names begin with an underscore ”\_”
- a value is a string or a DML expression enclosed in square brackets
- value strings are arbitrary ASCII strings enclosed in double quotes
- quotes may be omitted, if the value string does not include whitespaces nor square brackets
- value strings may be regular expressions (special characters escaped by a backslash)

# DML schema format

---

- a schema specifies the format of an attribute tree by specifying key names, attribute value types and the number of attribute instances
  - %S - String attribute with any number of instances
  - %I1 - Integer attribute with at least one instance
  - %F2! - Float attribute with exactly two instances
  - %S2<5 - String attribute with 2, 3, 4 or 5 instances
  - %T1:.key1.key2 - Internal attribute with at least 1 instance
  - %S<2:[1-9][0-9]\*[ \t]+[GMkK]B - String with at most 2 instances matching the regular expression (e.g. 125 kB)

# Reserved DML attributes

---

- **\_schema**: points to a DML tuple, which defines the schema of an attribute
- **\_find**: is replaced by the DML tuple specified (has-a relation)
- **\_extends**: make this attribute inherit all attributes in the specified path (is-a relation)

# Attribute ordering

---

- the order of attributes of a different name is insignificant
- the order of attributes with the same name is significant, e.g. the order of `ProtocolSession` attributes can be used to determine the order of protocol initialization

## Example:

```
graph [  
    ProtocolSession[name server use test.masterServer port 10]  
    ProtocolSession[name socket use SSF.OS.Socket.socketMaster]  
    ProtocolSession[name tcp use SSF.OS.TCP.tcpSessionMaster]  
    ProtocolSession[name ip use SSF.OS.IP]  
]
```

# Further information

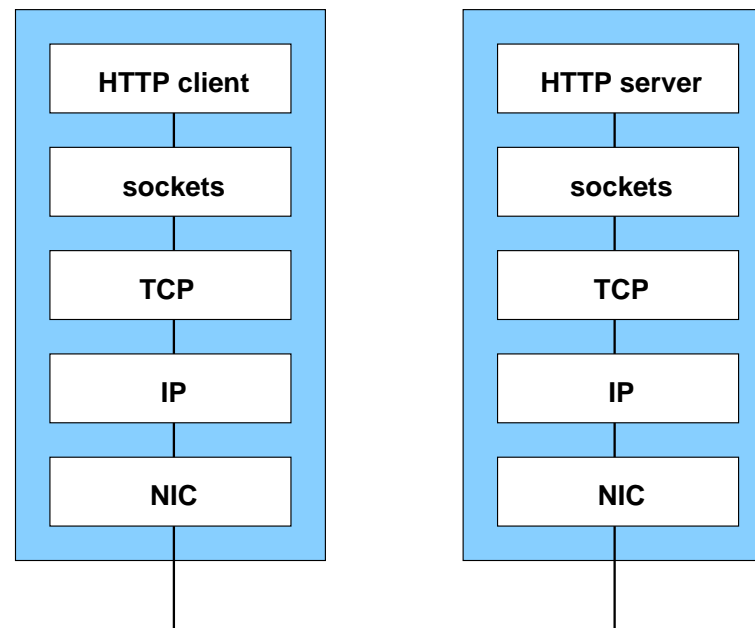
---

- Information about SSFnet can be obtained from:  
*<http://www.ssfnet.org>*
- Information about the Dartmouth SSF implementation:  
*<http://www.cs.dartmouth.edu/research/DaSSF>*
- BGP and OSPF implementations in SSFnet:  
*<http://www.cs.dartmouth.edu/~beej/research>*
- These slides: *<http://www.d-jacob.net/diplom>*

# Example 1: Peer-to-peer network

---

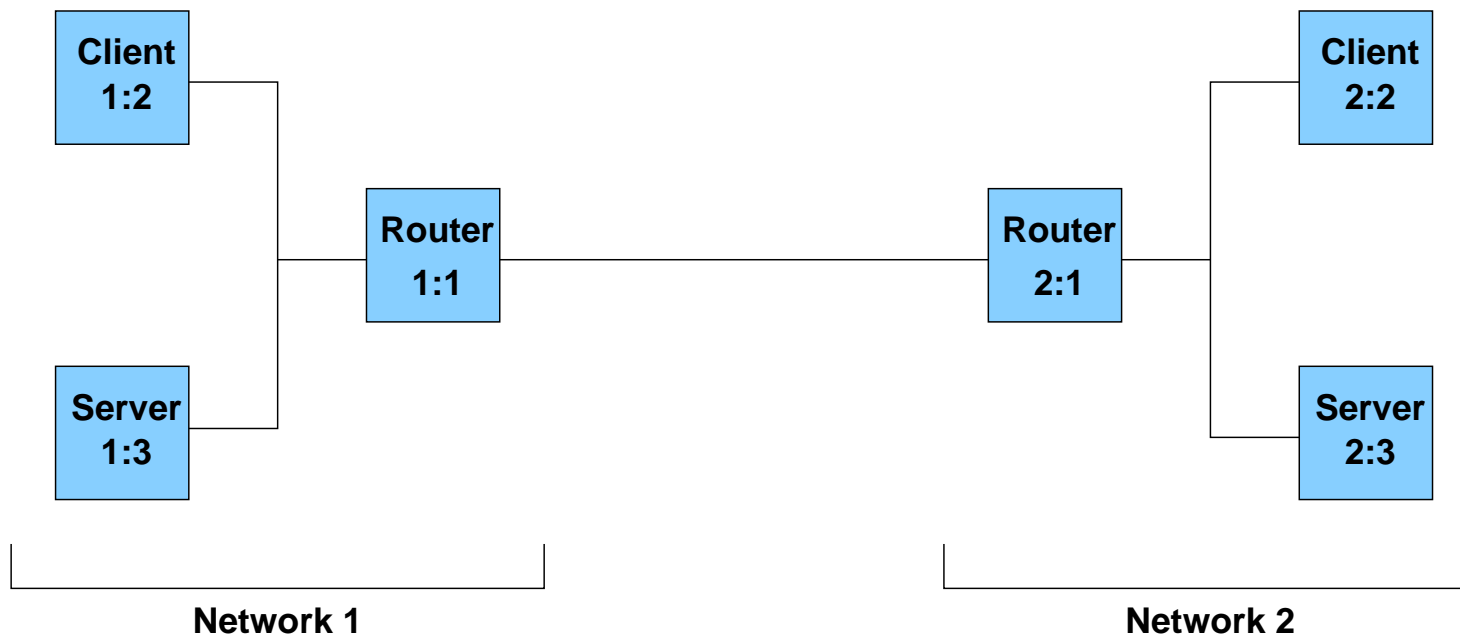
Topology:



# Example 2: Simple AS with routing

---

Topology:



# Example 3: Two AS's with BGP and OSPF

---

Topology:

