Message Groups

Dave Otway

History

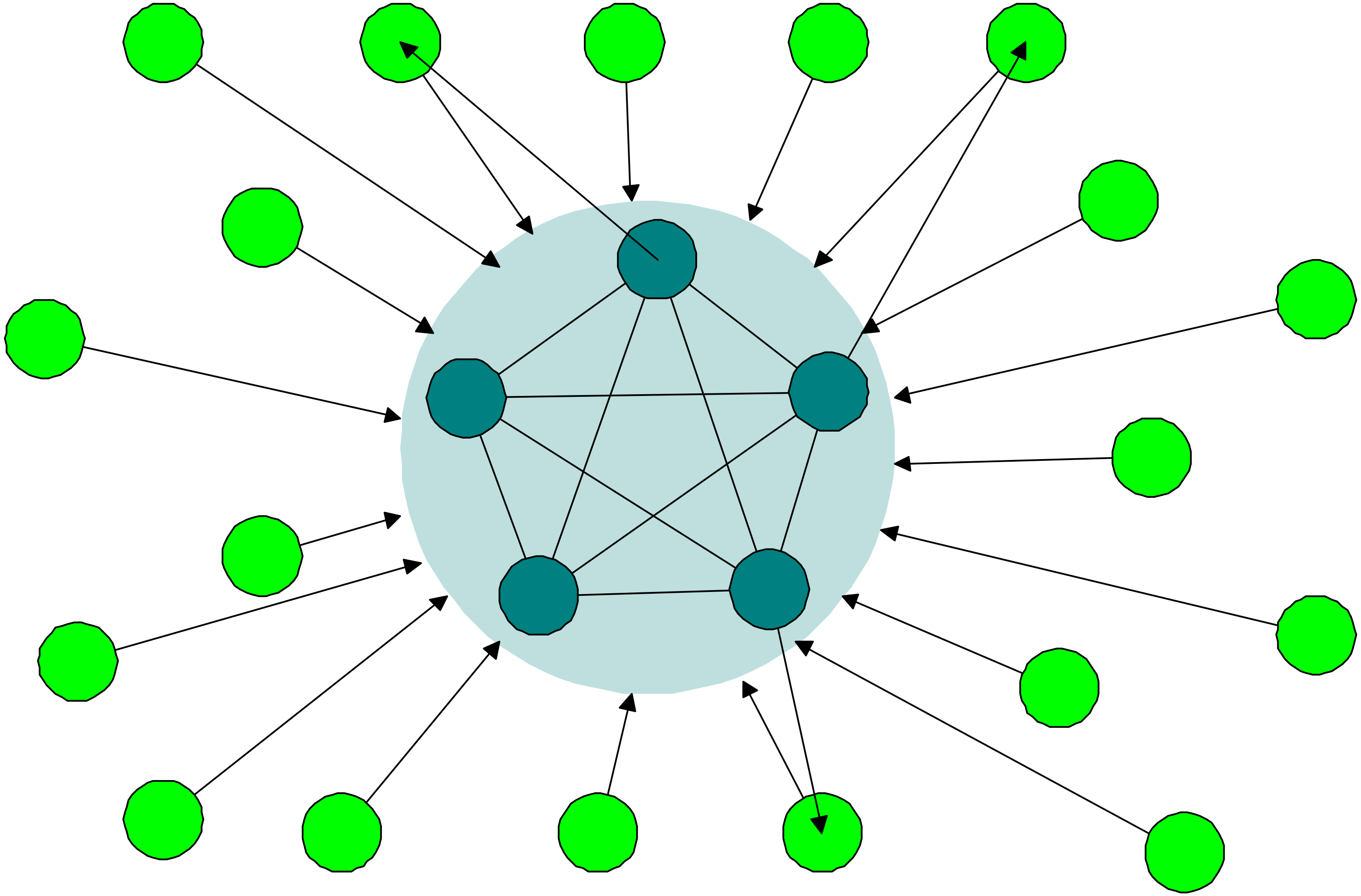
- FlexiNet - none
- Ansaware - GEX
  - RPC process groups
  - Chang and Maxemchuk protocol
  - multiple RPCs
- problems
  - collating multiple replies
  - inter group invocations
  - multiple RPCs
  - fragmentation

Purpose

- provide an infrastructure component for a distributed management architecture for FlexiNet
- FlexiNet is very "programmer friendly" but not easy for an uninitiated user to configure and manage
- management requires a consistent reliable view of a very flexible system

A Management Database

- all systems contribute data
- any system may perform management tasks
- size of database is not very large
  - size of dynamically changing data is very small
  - latency requirements are seconds not milliseconds
- resource usage more important than performance
- must scale well [from 2 to 1000's of systems]
- consistency and resilience are crucial



Multicast Groups

process groups

provide reliable processing by redundantly executing the same method in a small number of processes

usually used for high throughput transactions

message groups

provide reliable transmission and consistent ordering of messages

so that they can be resiliently stored by multiple processes

Multicast Group Protocols

Totem

a sender must be a member and hold the token

latency and buffering are proportional to the number of users

flow control is an integral part of the protocol

RMP

non-members can send messages to and interrogate the group

latency and buffering are proportional to the number of full members

flow control is orthogonal

Reliable Multicast Protocol

http://research.ivv.nasa.gov/RMP/

- based on Chang and Maxemchuk
  - + an ACK can order multiple messages
  - + an optimized orderly membership change protocol
- variable QoS
  - unordered, source ordered and totally ordered
  - k resilient, majority resilient and totally resilient
- both peer-to-peer and client/server models
  - client/server messages and RPC
- flow and congestion control
- non-multicast capable members

Sub-protocols

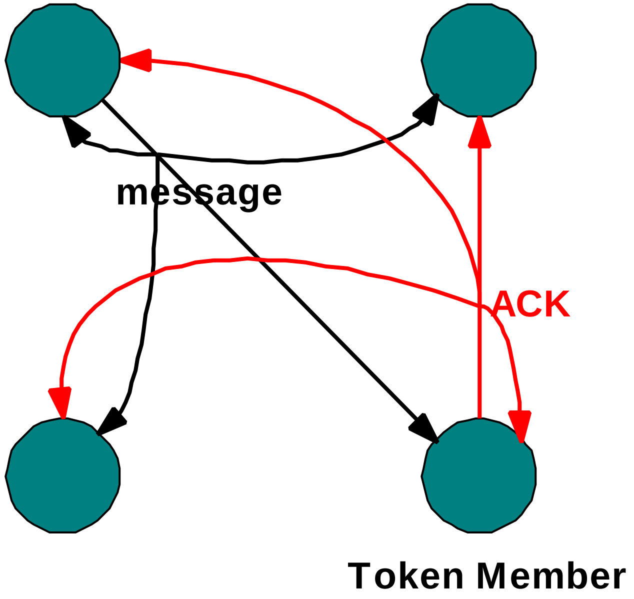
- basic delivery
  - including multicasts from non-members
- fault recovery - group reformation
- orderly membership change
- multicast RPC - from non-member to group
  - multicast request, unicast reply from one member
- flow and congestion control
  - modified sliding window based on Van Jacobson [TCP]

Basic Delivery Ordering

each message is given a total order by a single ACK from a rotating token member

the ACK also passes on the token to the next member in the ring of members

when no messages, the token is circulated once round the ring on timeouts

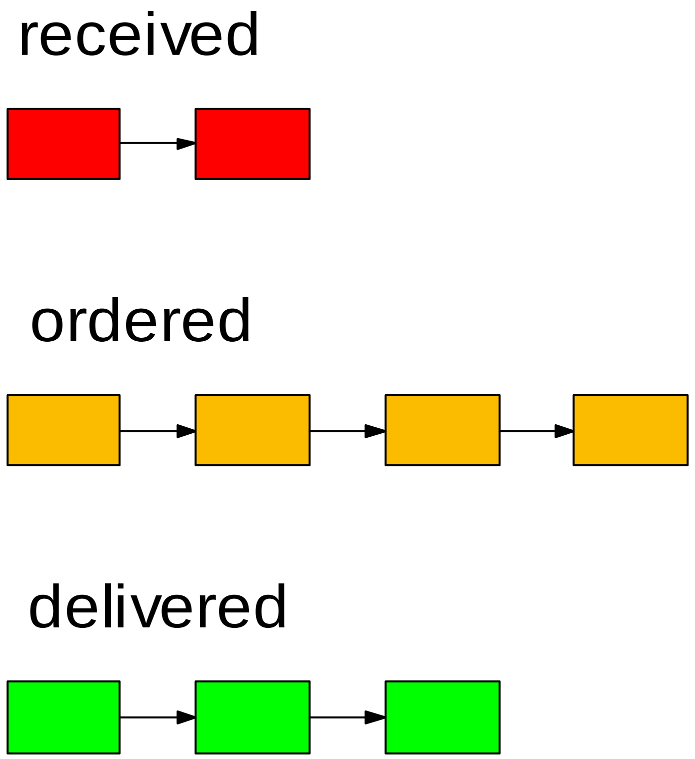


Basic Delivery Reliability

missing messages are requested by a NAK to the previous token member

a member will not accept the token until it has received all messages up to the one acknowledged by the token

a member must store messages for one complete token circulation



Basic Delivery Resilience

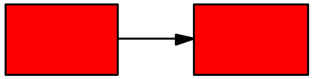
for total resilience,

messages aren't delivered until they have been ACKed by each member

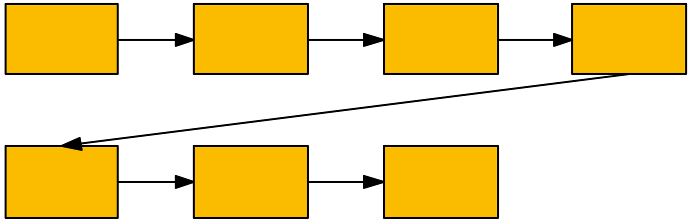
i.e. the token has done a complete circulation after they were ordered

they can then be discarded as soon as they have been delivered

received



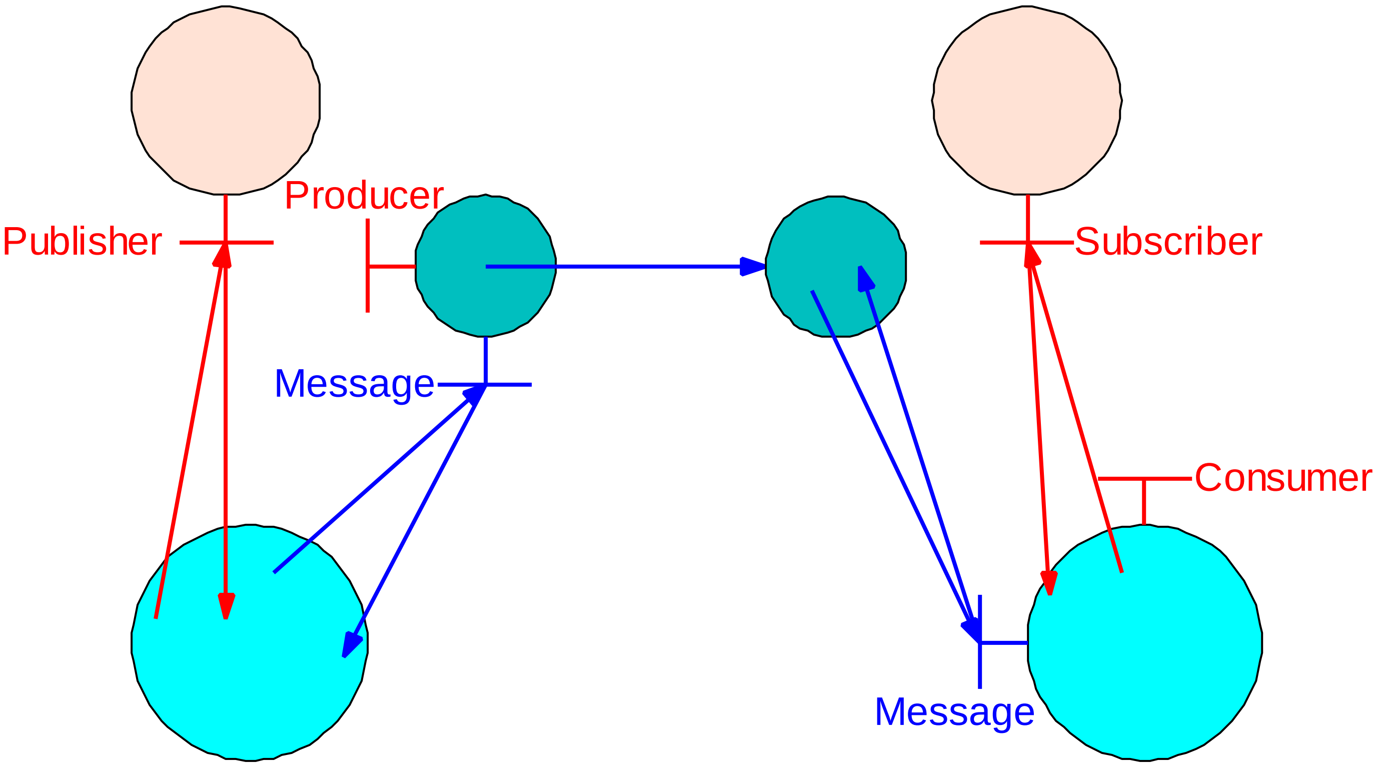
ordered



MessageGroup API

- Type safety
- Transparency
  - confine knowledge of group protocol and management
- flexibility
  - variable participation
  - choice of protocols for different QoS requirements
- usability
  - minimise application involvement in group management
  - make constructing group members easy and natural

Simple Producer / Consumer



User Interfaces and Classes

MessageInterface *E.g. Foo*

Defines names and types of application messages (and interrogations) that can be sent, delivered to (and invoked on) the group

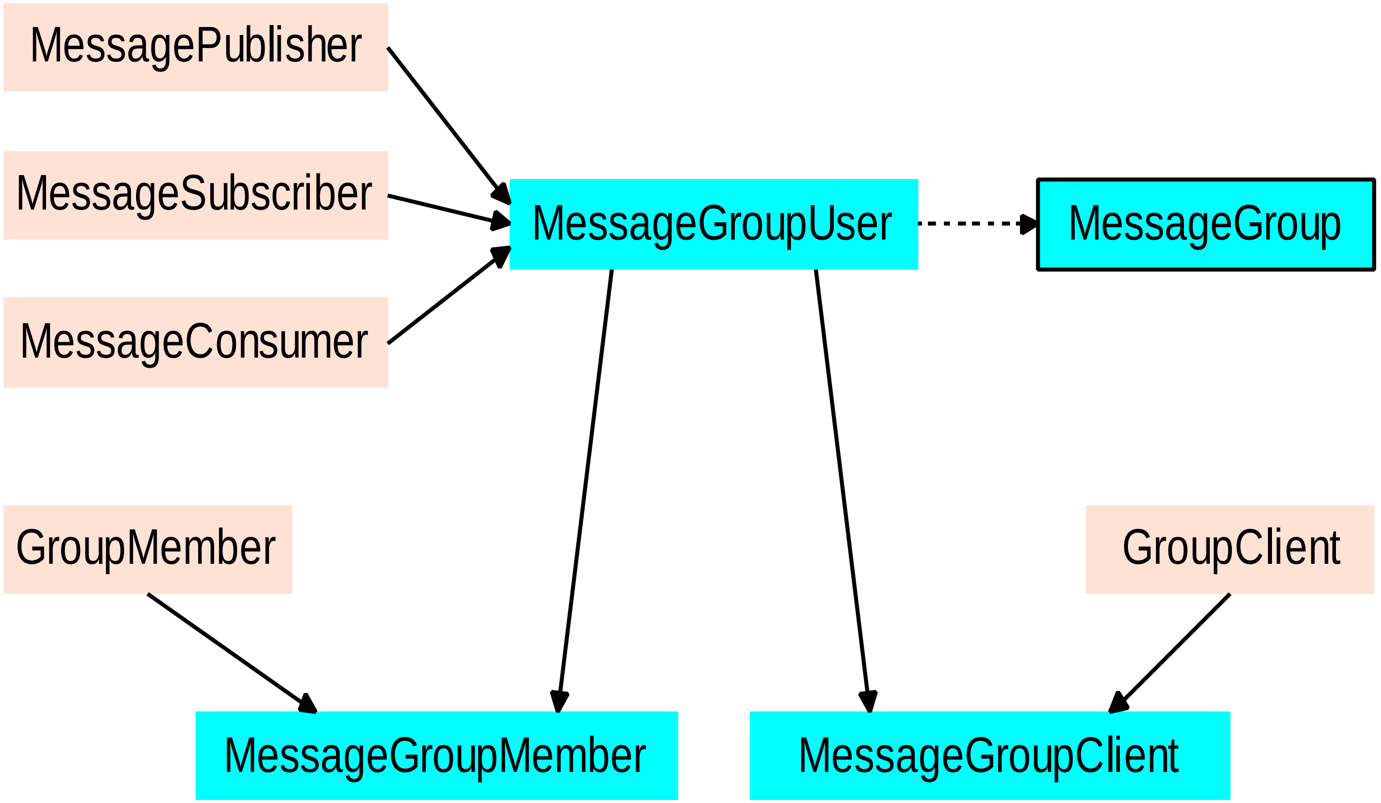
[messages return void and return early]

- Consumers provide a class which implements MessageConsumer and the group's MessageInterface
- Producers publish the group's MessageInterface, which generates a client stub implementing MessageProducer and the group's MessageInterface

Management Interfaces

MessageConsumer	*Manages ordering and single-threaded message delivery*
MessageSubscriber	*Binds/unbinds Consumers*
MessageProducer	*Manages the sending of messages*
MessagePublisher	*Generates Producers and binds/unbinds them*
GroupMember	*Consumes and stores all messages (may produce)*
GroupClient	*Produces and/or consumes messages (may interrogate)*

Management Classes



Consumer Implementation
[class FooMember implements Foo]

- Inherited from MessageGroupMember
  - simplest - inherits all management
  - but looses transparency (like RMI)

- Composed from MessageGroupMember
  - need to implement or indirect management methods
  - can inherit from a non MessageConsumer class

Constructing a Foo Group

first member

- fooMember = new FooMember()
- fooProducer = fooMember.publish(Foo.class)
- fooGroup = fooMember.group()
- pass fooGroup to prospective members

other members

- fooMember = new FooMember(fooGroup)
- fooProducer = fooMember.publish(Foo.class)

Publishing a Foo Client

any member

- pass fooGroup to prospective clients

clients

- fooClient   = new FooClient(fooGroup)
- fooProducer = fooClient.publish(Foo.class)

or

any member

- pass fooProducer to prospective clients

clients

-

[FlexiNet does new FooClient(fooGroup) when binding]