# IDAWI a P2P middleware for distributed computing

Luc Hogie

*luc.hogie@cnrs.fr*
*I3S Computer Science laboratory, Inria, Université Côte d'Azur, France*

January 10, 2021

IDAWI s an open source middleware for distributed/parallel applications. Its most significant departure from other tools is its completely decentralized P2P architecture. The design of IDAWI s driven by the multiple use cases for scientific and engineering projects we have in the lab, and it benefits from the experience we acquired in past projects. For many years we have been studying distributed systems from both theoretical and practical perspectives 2) . Also we have been users of technologies for distribution as weconducting experiments on data sets which are so huge that they not have been processed without resorting to distributed computing. , its existing software solutions, and to face their limitations. Indeed each new experiment came with its peculiar data model and specific requirements, and existing frameworks turned out to be inadequate. Each time we had to write significant amounts of code to have suitable software solutions to our problems. In this context IDAWI an be seen as a synthesis of these past developments and of our experience in the field distributed computing. As such it gathers implementations of the concepts that proved useful to our past studies, and that we believe will be useful to future Research not only useful in our local Research groups but to a broader community.

Distributed computing is notoriously difficult field. Writing distributed applications is hard as programmers have to deal with complexities at all stages: heterogeneous hardware and networks technologies, sophisticated protocols, complex software infrastructures, concurrent access to data, performance issues, reliability issues, etc. IDAWI ackles this issues as a whole by using an elegant approach which proved highly successful when used as the core of world-scale P2P infrastructures: decentralization. IDAWI efines a dynamic graph in which nodes can rely only on their local view and on their interactions with their changing neighborhood, without relying on any centralized service. Decentralization endows IDAWI n elastic nature that has numerous intrinsic advantages, such as the ability to grow or shrink dynamically, to be resilient to failures, to accommodate to node mobility and churn. This makes Idawi uitable to a wide range of applications like IOT, MANETs, cluster and cloud computing, etc.

On top of this decentralized model, Idawi roposes the following features, among others:

- automatized deployment of nodes through SSH
- both asynchronous and synchronous messaging
- unicast, multicast, and broadcast communication
- streaming
- scalable
- massive multi-core parallelism
- multi-protocol: support SSH, TCP, UDP, etc
- REST-based web interface

The design objectives of Idawi re to make it scalable/efficient, easy to use, elastic, versatile. Also, in order to be usable on the largest variety of devices, ranging from IOT sensors to super-calculators, Idawi s written in Java.

## 1 Component

Idawi efines an overlay network of components. Components can be physically located within a same JVM, in different JVMs on the same computer of in remote computers.

A component can be referred to in the graph by its ID.It essentially is a container of services.

## 2 Service

Components have no application logic. Application code is held by *service*s exposed by components. A service is the atomic application element in a Idawi istributed application.

A service provides implementation of code targeting a specific concern. Generic concerns like routing, error management, deployment, benching, are already tackled by builtin services. Likewise, an new user application will implement its own specific concerns as one or several services.

## 3 Operation

A service is organized as set of operations. An operation implements a specific action in a given service. Operations are exposed to other services in the host component as well as to other components in the network.

The code of an operation is triggered upon the reception of a certain type of message. More precisely when such a message is received, the corresponding operation is scheduled to be executed by a pool of threads. This execution will happen as soon as possible, possibly in parallel to the execution of other operations of any other or same service/component in the system.

Unlike Java methods which cannot return any value to their called before the end of their execution (using the *return* keyword), IDAWI perations can return information anytime. This mechanisms allows several applications, such as remote monitoring. This return information is carried by messages which do not aim at triggering operation, but at delivering information into a specific *queue* previously created by the caller of the operation.

### 3.1 Declaring an operation

In the user code, operations can be declared in many ways: as methods, as fields containing an implementation of a functional interface (which are especially relevant when they are written as lambdas), as serializable fields (which entails the automatic creation of a *getter* operation) or programmatically. Every single way of declaring an operation has its own advantage.

### 3.2 Parameterized operations

As-method operation enables *parameterized* operations. Operations input data is specified by the parameters of the representative method.

### 3.3 Dynamic operations

Programmatic declaration enables *dynamic* operations. Operations can be created/deleted at runtime.

## 4 Descriptors

A component descriptor contains information about a component at a certain date. The set of descriptors held by a component constitutes the local knowledge of the system. If changes occur in the system (addition/removal of components/services/operations, network conditions, computer loads), this local knowledge tends to get outdated. For this reason, components periodically produce up-to-date descriptors of themselves and disseminate them across the system.

Also descriptors are fed by neighborhood information carried by messages as they travel in the graph.

Note that a component descriptor may refer to a non-existing component.

Nodes have a local registry which stores information about their peers. This information is not necessarily complete for a given peer nor it The *neighborhood* of a node is the set of peers. Node incorporate a routing service, which enables nodes to interact with peers not in their neighborhood by using neighbors as relays.

## 5 On scalability

This graph of component does not have constraints regarding its dimensions. It can be as small as the singleton graph, consisting of one single component, or made of huge amount of components. As a consequence, it cannot be ensured that components have a complete knowledge of the graph: the larger the graph, the less accurate the *local view of components.*

*This local view consists of a set of descriptors for the components in the graph.*

## 6 Deployment

*A unique feature of* IDAWI *s its ability to deploying new nodes programmatically. More precisely, if a node (called the parent node) is given the ability to connect through SSH to a remote host, it becomes able to start new nodes (called the child nodes) on this host. To do that, the parent node uses the ubiquitous* `rsync` *utility to incrementally transfer all its binaries (the whole content of its Java classpath) to the remote host. Once the binaries transferred , the parent node starts a new Java Virtual Machine on the remote host, with an instance of the node class in it. This new node will instantly being in capacity to communicate with its parent node, as well as with other peers in the overlay.*

*In practise, instead of deploying one single node to one single host at a time, a parent node will rather try to deploy new nodes onto multiple hosts in parallel. Here arises the problem of shared file-systems to which writing operations must deal with conflicting resources: before transferring binaries to remote hosts, the parent node executes a distributed algorithm for the computation of the set of hosts sharing a same file-system. Then binaries are transferred in parallel to one single (random) computer in every set. As soon as a set as received its binaries, child nodes are in parallel started on all hosts in it.*

*At any time, a child node can be set to be dependant on its parent node. In this case, child nodes terminates on their own as soon as they lose the SSH connection to their parent node. On the contrary an independent child node will continue running even so the SSH connected that were used to start it gets lost. Note that an independent node may have other ways to communicate with its parent if it is still running, thereby not losing contact.*

## 7 Message

*A message is identified by a unique random 64-bit numerical ID.* IDAWI *essages transfer is based on multicast/broadcast, meaning that messages are targeted to not a single peer but to a set of peers. Unicast can be emulated by specifying one single element in the set of recipient peers. It carries a content which can be any Java object.*

*Analogously to e-mail messages, messages have a* to: *as well as an optional* reply-to: *address.*

*An address is defined by a 5-uplet* $(C, s, o, e, p)$ *where*

**C** *is the set of component IDs (messaging in* IDAWI *s multicast by nature). If C is not set, the address is considered to be a broadcast address*

**s** *is the identifier of a service on all target components*

**o** *is the identifier of an operation on all target services*

**e** *is the maximum number of hops allowed*

**p** *is the forward probability*

*After a message has been processed by a component, it is not dropped. Instead, until it expires, it is stored and considered for re-emission each time a new neighbor component pops up. This enables* IDAWI *o deal with node mobility and scarce connectivity found for example in delay tolerant networks (DTNs).*

## 8 Asynchronous communication

*The base primitive for sending messages is* `send(msg, to, reply-to)` *where msg is the message to send, to is the destination address and replyTo is the optional address for sending results.*

*Messages is sent asynchronously. This primitive is not blocking and the emitter gets no acknowledgement of reception.*

*If the* reply-to: *is undefined, the target operation has no mean to send anything, and in particular to send returns back to its caller.*

*If defined, the* reply-to: *address is not related to the* to: *address: the target operation may send results to any address.*

*When using the* `send(msg, to)` *primitive, prior to issuing the message the emitter service 1) sets a default* reply-to: *address back to the emitter, and 2) creates a local message queue dedicated to the reception of eventual return information.*

## 9 Synchronous communication

*A message queue is analogous to a* future. *It defines a* `get(timeout)` *primitive which blocks until a message is available or the specified timeout expires. By* *invoking this primitive, the emitter service does synchronous communication. If set to a non-infinite value, the timeout ensures no dead-locks will occur in the system.*

## 10 Underlying transport protocols

IDAWI *eatures an abstraction for the network communications. It defines a protocol driver as a component able to send messages asynchronously, to trigger a consumer each time a new message is received, and to provides a set of peer in its current neighborhood. The following protocols are currently supported.*

*LMI is used when the two components involved in the communication are in the same JVM.*

*UDP is the fast IP-based transport layer. Lack of ACK is compensated by* IDAWI *ynchronous communications.*

*TCP is slower than UDP but has two significant advantages. 1) it imposes no constraint on the size of messages 2) it permits SSH-tunneling, which is required when the plain TCP ports are inacessible because of a firewall or a NAS.*

*SSH is primarily used to deploy components on remote computers. The communication streams between the SSH client and SSH server are used to transport messages. This mechanism can be used even when SSH tunelling not available.*

*Sometimes nodes have no direct neither indirect way to reach one another using other protocols, but they share a common file system. This is the case of two computers in different LANs, each one connected to a common cloud storage.*

## 11 Multi-thread parallelism

*Some systems like ProActive define that every entity (active objects in the case of ProActive) have their own thread. This appealing model has one major drawback: as the number of threads is limited, the system cannot scale.*

*In* IDAWI *components have no threads associated to them. Their executable parts (operations) are scheduled to a pool of threads shared by all components into a JVM. By default, there is only one JVM per computer and as many threads as cores on the local computer.*

### 11.1 Service lifecycle

*Thanks to the service manager service, services can dynamically start and stop other services in their node using an object-oriented API, as well as in other nodes, using a message-based API.*

## 12 Fitting to experimentation

*R&D and experimentation have their own requirements. In particular, designing and tuning a distributed algorithm often is a* trial and error *process. As such, it is crucial that the developer is able to perform runs very quickly and to get useful report of errors.*

### 12.1 trials...

*In order to allow quick trial runs,* IDAWI *eatures automatic deployment of the local code. More precisely,* IDAWI *utomatically transfers to the remote hosts the whole content of the local Java classpath. Then, Java code and resources get transparently deploy across the network, be they located in a project directory for an IDE, or packaged in a Jar file. This constitutes a significant departure from other middleware which usually impose the programmer to deploy its code by hand.*

### 12.2 errors...

*In order to report errors to the programmer,* IDAWI*) forwards back the standard output and standard error streams of remote virtual machines, and 2) it features a specific distributed service dedicated to logging and disseminating errors, which allow any component in the system to identify faulty code.*

## 13 REST interface

IDAWI *omponents come with a specific service dedicated to monitoring the system. This services starts a REST-compliant HTTP server which make all components/services/operations in the system accessible from a web browser.*

*The Web interface uses Bootstrap and Vis.js on top of HTML 5 so as to propose modern scientific views of the graph.*

## 14 Comparison to existing platforms

*https://dl.acm.org/doi/10.1109/TSE.2015.2476797*

*ZeroMQ*

*Monix*

*Zio*

*JGroups*

*Akka*

*Hazelcasts*

*Terracota*

*Apache River*

*Jade is supported by Telecom Italia. Its naming service is centralized In accordance to the FIPA standard, Jade imposes a number of component-specific* *high-level asbractions which concern their behavior, their interactions, etc. It receives messages in blocking mode.*

*Unique features of* IDAWI *re P2P, deployment/*