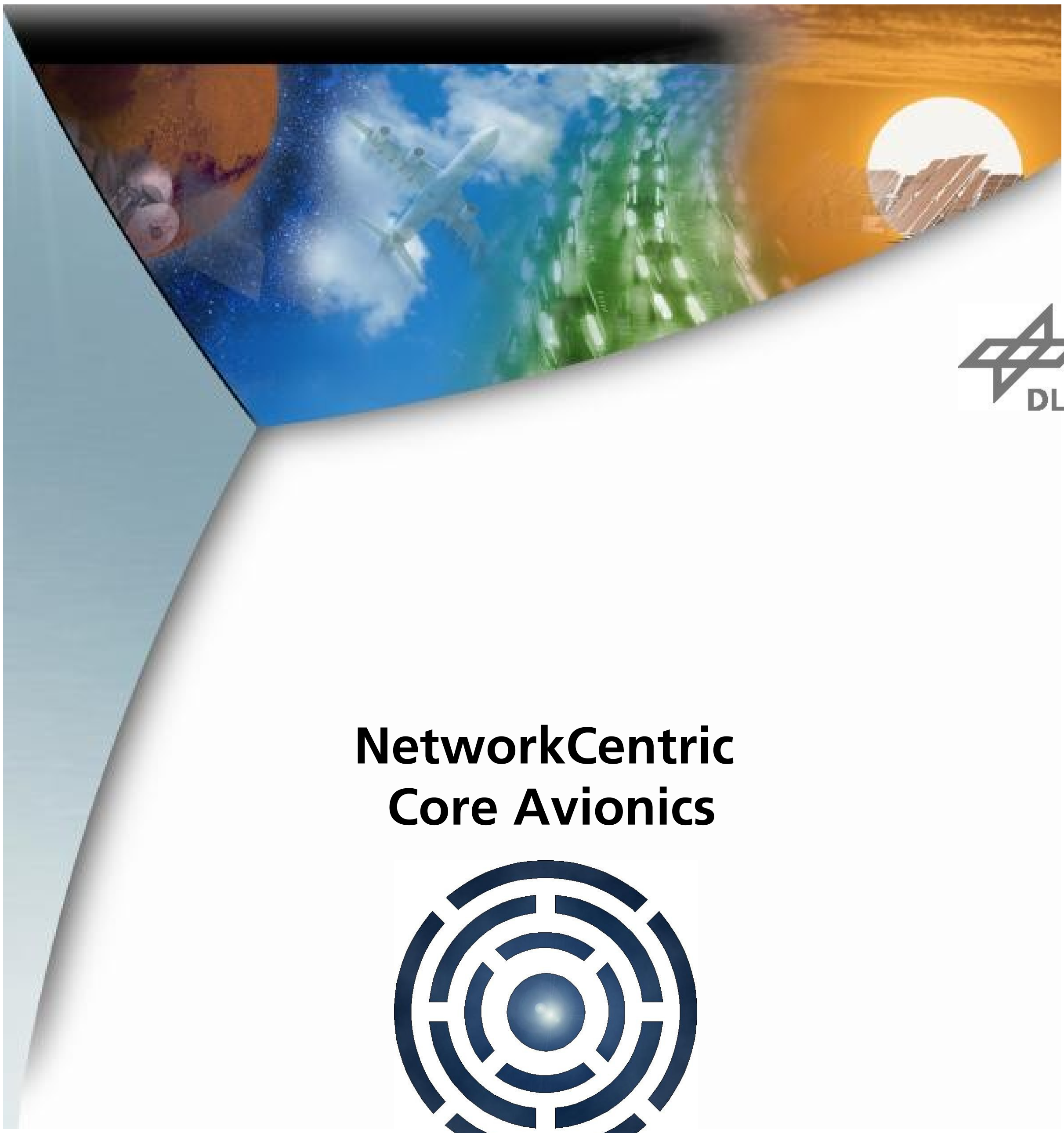


NetworkCentric Core Avionics

RODOS Tutorial Task Distribution

Version: 1.0
Date: 10.10.2010
Author: Sergio Montenegro

RODOS Tutorial Task Distribution

The task distribution tutorial shows an example of how to distribute tasks in different computer nodes and how to control, on which computer node a task shall be executed.

This strategy can be used for both: fault tolerance and load balancing.

The job of the system is performed by "user tasks", which are supported by task managers and a task distributor.

There may be different types of user tasks.

Tasks of the same type (on different nodes) interchange their context periodically, to allow a sleeping task to start working with the most recent context.

Each node has a local task manager which controls which tasks shall be active and which shall be sleeping (not active) in the local node. The local task manager periodically sends state information of the own node to a central task distributor.

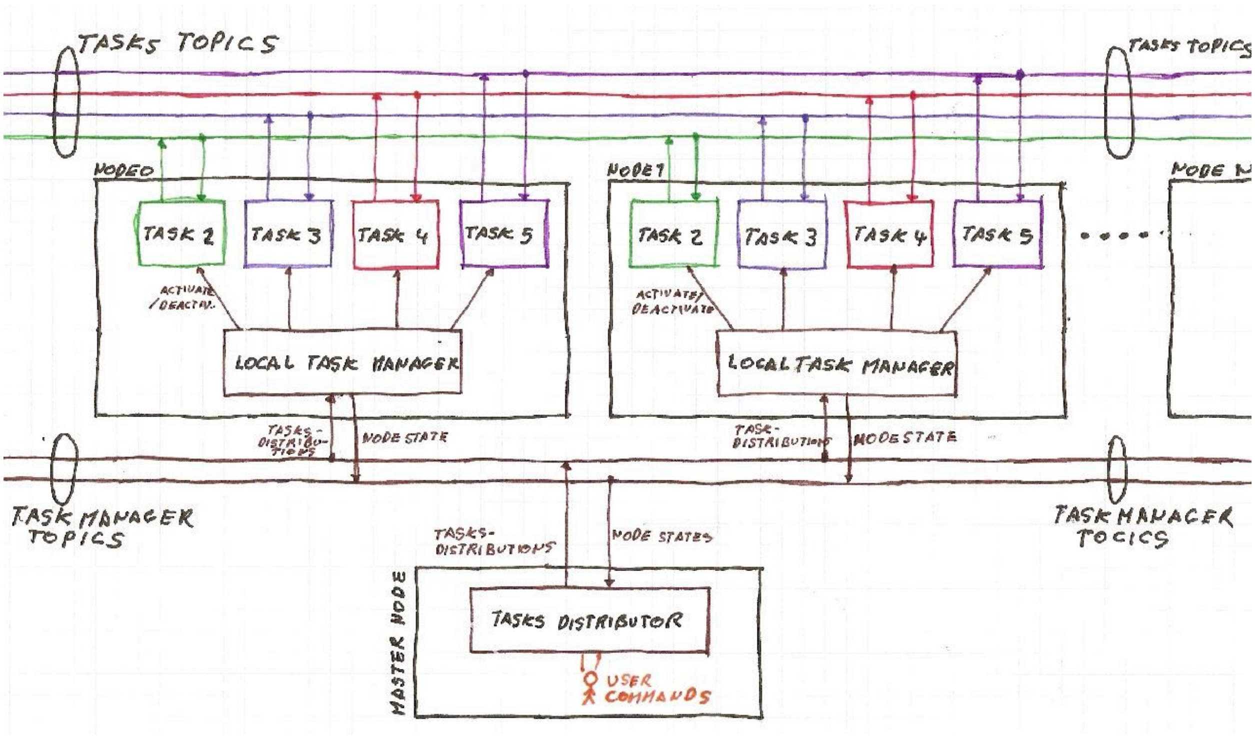
The task distributor decides on which node which tasks shall be executed. If a node crashes, the distributor will command local task managers to migrate the tasks of the crashed node. To migrate a task X from node A to B means to deactivate (set to sleep) the task X in node A and activate another instance of the task X task in node B. If the node A fails, then we just have to activate task X in B.

The task distributor may be executed in any node, together with other tasks, but for this example it will be executed on its own node, which is called the master node. A more elaborated implementation could distribute and/or replicate the task distributor too, but for this example, we take a simple solution.

All nodes shall have a local task manager and all user tasks loaded. The number of nodes is not fixed. Each node (more exactly: the local task manager on each node) will report it's state to the task distributor. So the task distributor knows which nodes are ready to execute tasks.

The task distributor creates a task distribution table and distributes it to all local task managers. Each local task manager can see on this table which tasks shall be active in it's node.

Overview



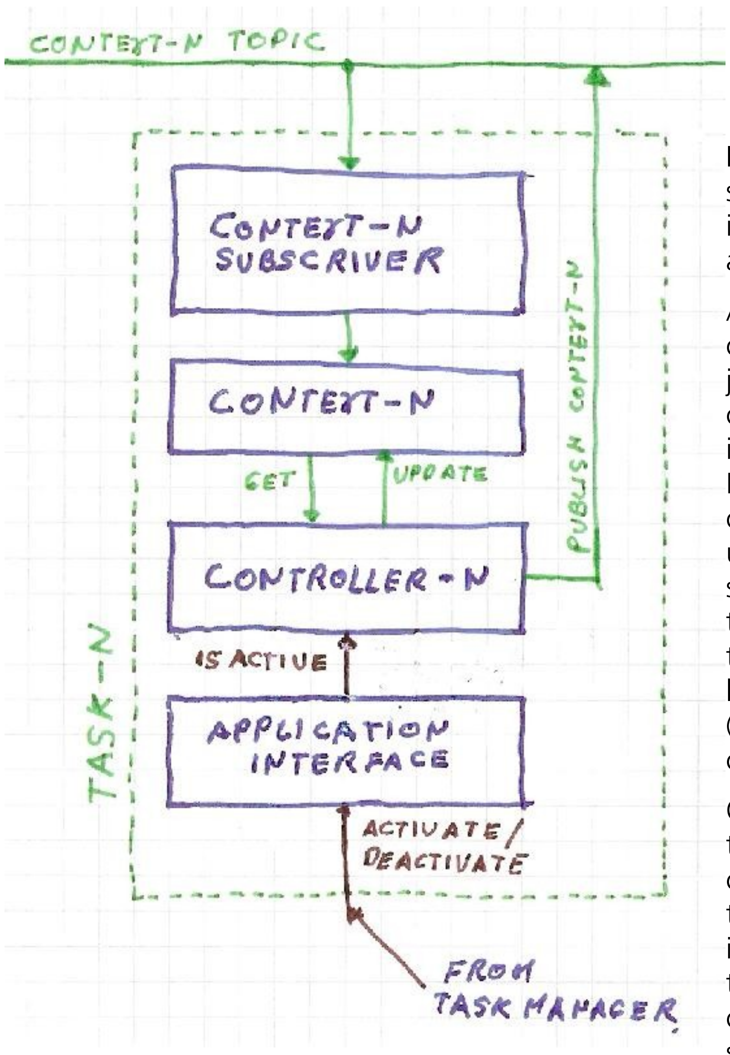
Each node has exactly one instance of each user task type. All tasks of the same type (distributed in several nodes) interchange their context using a dedicated topic of the middleware.

The local task manager uses the application discovery protocol to find the loaded task in it's own node, and to access the application interface of the loaded tasks. The application interface provides activate() and deactivate() methods.

The local task managers hear (subscribers) from the task distribution topic to get the distribution table to know which tasks shall be active and which shall be sleeping in their node.

The local task manager reports the state of it's node using the node state topic. The task distributor hears (subscriber) messages from the node state topic to know which nodes are available. Using this informations it builds the task distribution table and distributes this table using the topic task distribution.

User tasks

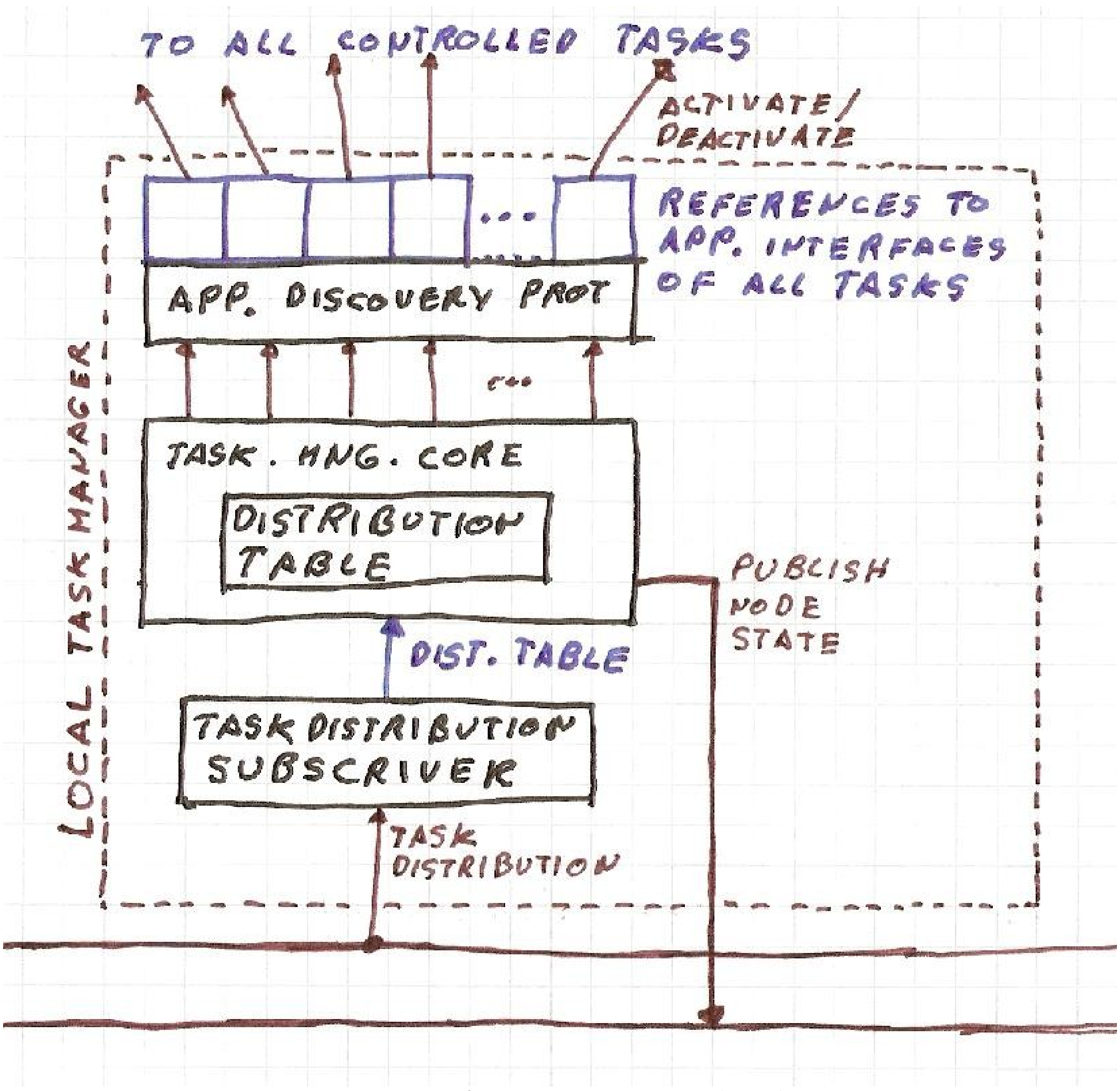


In this example all user tasks have the same structure (to make it simpler) which is recommended as starting point for real applications.

A central point of each user task is it's context. In the examples the context is just a simple counter (long long). The context is used by the controller – when it is active – to compute a behaviour. Each time the controller modifies the context it distributes the updated version using the corresponding topic. The subscriber of all other tasks of the same type (on other nodes) will get a copy of the current context and update their local context. So when a task is activated (in any node) it can take the most recent context to continue operations.

On the lower part of each task, we have the application interface which gets commands – activate / deactivate – from the local task manager. The application interface just sets a flag to indicate if the task shall be active or sleeping. The controller checks this flag to know if it shall work or sleep.

Local Task Manager

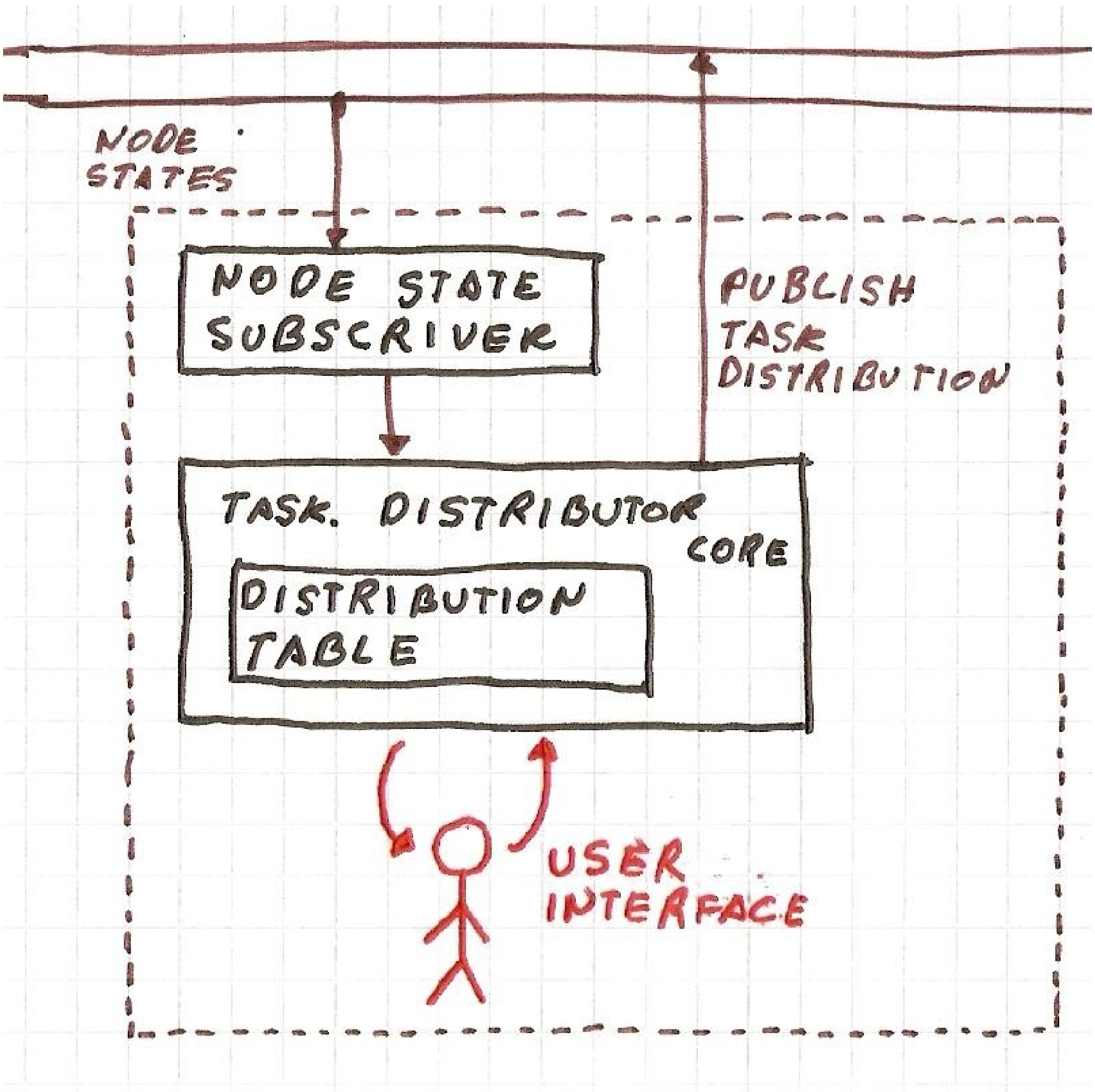


The local task manager creates a list of loaded applications (tasks) on the local node. For this purpose it uses the standard application discovery protocol of the RODOS application interface. The task manager uses this list to send activate/deactivate commands to each application using the standard application interface.

The task manager has a subscriber which gets the task distribution table from the task distribution topic. The subscriber updates the local distribution table. The task manager core checks this table to activate the tasks which shall be running in the current node and deactivate all other tasks.

The task manager core periodically collects state information of the local node, In this example just an indicator for CPU load. Other possibilities could be: free memory, list of loaded tasks, etc. The task manager publishes the node state using the node state topic.

Task distributor



The task distributor has a subscriber from the node state topic, to know which nodes are usable. In this example all nodes are supposed to be identical, in more elaborated implementations nodes could have a different task list and may have different performance, they could have different devices attached, which would be reflected in the list of loaded tasks (interfaces to devices).

Knowing which nodes are available and executing user commands, the task distributor core creates a task distribution table which says, for each task, on which node it shall be executed. The task distributor distributes (publishes) this table periodically using the task distribution topic.

Content of the example

In this directory you will find:

taskmanager.cpp

The local task manager. It subscribes the topic topicTaskDistribution and publishes the topic topicNodeState.

taskdistributor.cpp

The master task distributor. It subscribes the topic topicNodeState and publishes the topic topicTaskDistribution.

usertask_negCnt.cpp

An example of a user task. It counts negative integer number. It publishes and subscribes it's context using the topic topicNegCnt.

usertask_oddCnt.cpp

An example of a user task. It counts odd integer numbers. It publishes and subscribes it's context using the topic topicOddCnt.

usertask_evenCnt.cpp

An example of a user task. It counts even integer numbers. It publishes and subscribes it's context using the topic topicEvenCnt.

usertask_primCnt.cpp

An example of a user task. It finds prime numbers. It publishes and subscribes it's context using the topic topicPrimNums.

globals.h and **globals.cpp**

Definition of all tasks-ids, context classes and topics in the system. All common definitions for tasks and managers.

How to execute

To execute (on linux) first create the linux lib:
Call in the rodos root directory

```
% source make/rodosenv
% linux-lib
```

Compile the network simulator
in the directory support_rograms

```
% doit-linux
```

In the directory where this example is stored (tutorials/taskdistribution) use the ready to use shell script

```
% executeit
```

This script will create a user boot image (an executable) for the user nodes and a distributor boot image for the master node.

The user boot image contains all user applications and the task manager.

The distributor boot image for the master node which contains only the task distributor.

Then the script will create 4 xterm windows, one for each user node, one xterm window for the task distributor and one xterm window for the network simulator.

Note: The taskdistributor (yellow window) will wait until you type <cr> to start working.

Each xterm window will start the corresponding image.

Try closing user windows and see how the distributor migrates the tasks in the still running nodes.

Try typing commands to the distributor. Syntax:

<application-Nr> <node-Nr>

no names, no commas, just two numbers separated by blank. The first number is the application number (from 2 to 5), the second number is the node where it shall be executed. -1 means no node, task is nowhere active.

Die task distributor will distribute the task in such a way that the maximal load (number of tasks) of any node shall not be bigger than minimal load + 1.

If a node crashes (try control-C on any user node) the tasks from the crashed nod will be distributed on the oner nodes.

If a new node comes (create a new window and execute “tst_usernode”) the distributor will give it some tasks to execute.