## Distribution and Communication

Distribution and thus communication are two key aspects of scalability. In a very early version of the MARS system layers were only distributable as a whole, so each LayerContainer needed to take care of one ore more complete layers. Our findings however have shown, that one layer may be too complex for a single computer or we may have rather slow compute nodes (see **Fehler! Verweisquelle konnte nicht gefunden werden.**). So the new approach will also allow to distribute each layer across several LayerContainers, wich resembles true horizontal scalability.

Since distributing layers has direct influence on the agents living on them, our approach for layer distribution is tightly coupled with our approach to distribute agents and make the overall system scalable. That approach is called Agent Shadowing (Layer Shadowing respectively).

### Layer / Agent Shadowing

Assuming our architecture, where agents live in three dimensional layers each distributed across one or more container nodes, the problem of synchronization and communication arises, when it comes to agent interaction or movement across the boundaries.

Specifically whenever an agent wants to communicate (that is: call a method) with another agent, it first has to check whether the desired agent is present locally or remotely and, if remotely, obtain a reference through which it may perform the actual communication. By that each method call results in a (rather slow) remote message.

If agents move around their environment or are moved by a load balancing partitioning mechanism, it may well happen, that an agent crosses the virtual border of a container node’s part of the layer and thus has to be moved to another container node instance. If that happens, the communication reference of that agent has to be updated, whenever another agent holding an old reference wants to communicate.

### Concept

Agent Shadowing is the depiction of an agent living on layer A1 having its shadow drawn onto layer A2, where it is not actually instantiated, but instead is represented by a stub-like object as in remote communication concepts like RPC/RMI.

In RPC/RMI each agent’s methods are callable by third parties through its stub object. Usually a stub just provides the capabilities to establish an interface-bound communication with the remote object. If the remote reference changes, in classic RPC/RMI the stub simply becomes useless since its reference is not updated. The protocol then has to notice the broken link and re-establish a new one.

A shadow agent stub (SAS) is extended by the ability to hold cached attributes like its position or any other attribute. The real agent object updates both, the attributes and the remote reference, whenever a change occurs. These updates are delivered via multicast when in LAN to reduce the amount of traffic. The initial remote references can be provided when the overall system is initiated since some kind of distribution information has to be provided at that state.

This results in each container node containing the full environment as well as allThis results in each container node containing the full environment as well as all agents, but with the difference, that only numberOfAgents / numberOfNodes (given an even partitioning) agents are really instantiated (and thus have to be computed). The remaining agents are only instantiated as SASs and do not contain any agent behavior logic. An increase in container nodes would reduce the amount of agents per node that have to be actively computed, while the memory footprint per node would also potentially decrease, assuming that a SAS consumes less RAM than a full-fledged agent.

Calling or referencing another layer, works by the same pattern of either having a local instance of that layer to address directly or a stub to communicate with a remote reference (Layer Shadowing).

We postulate the following hypotheses for this approach:

1. This data-binding mechanism significantly reduces the amount of (duplicated) network communication / traffic between agents, because heavily used attributes may be cached in SAS.
2. Lookup of remote references is not necessary anymore, since each agent is virtually present at each container node and may be accessed through its usual interface, with the stub-object binding taking care of the remote reference.
3. Distribution of agents is transparent to the programmer.
4. No single-point-of-failure since no central directory for lookup or routing is necessary. Furthermore if a container node crashes, its state might be recreated by another node.
5. Massive traffic resulting from multiple simultaneous SAS updates, can be reduced by aggregating these updates into one large batch update.
6. The system is limited by the maximum amount of RAM per node .
7. This limitation can be compensated by introducing lazy loading of SASs, utilizing potential locality of agent interaction and a garbage collection for SASs which have been unused for too long.