LEAF

Learning Agent FIPA Compliant Community Toolkit

Steven Lynden, Omer F. Rana
Cardiff University, UK.

LEAF

Learning Agent FIPA Compliant Community Toolkit

interacting Java based learning agents existing in multiple communities to solve goals requiring collective behaviour – **communication**, **coordination**, **analysis**, **scalability**…

Plan of this talk

1. Agents, FIPA.
2. LEAF – concepts
3. FIPA-OS
4. LEAF – toolkit
5. Applications

LEAF Agents

- Autonomous
- Domain knowledge
- Learning behaviour
- Communication via Internet
- Interoperable (FIPA compliant)
- Research objective: deploy LEAF agents to perform goals that require collective effort, and support the scalability of this approach.

FIPA - Foundation for Intelligent Physical Agents

Standards for interoperable agent systems, to which developers adhere:

communication protocols

message transport

agent management

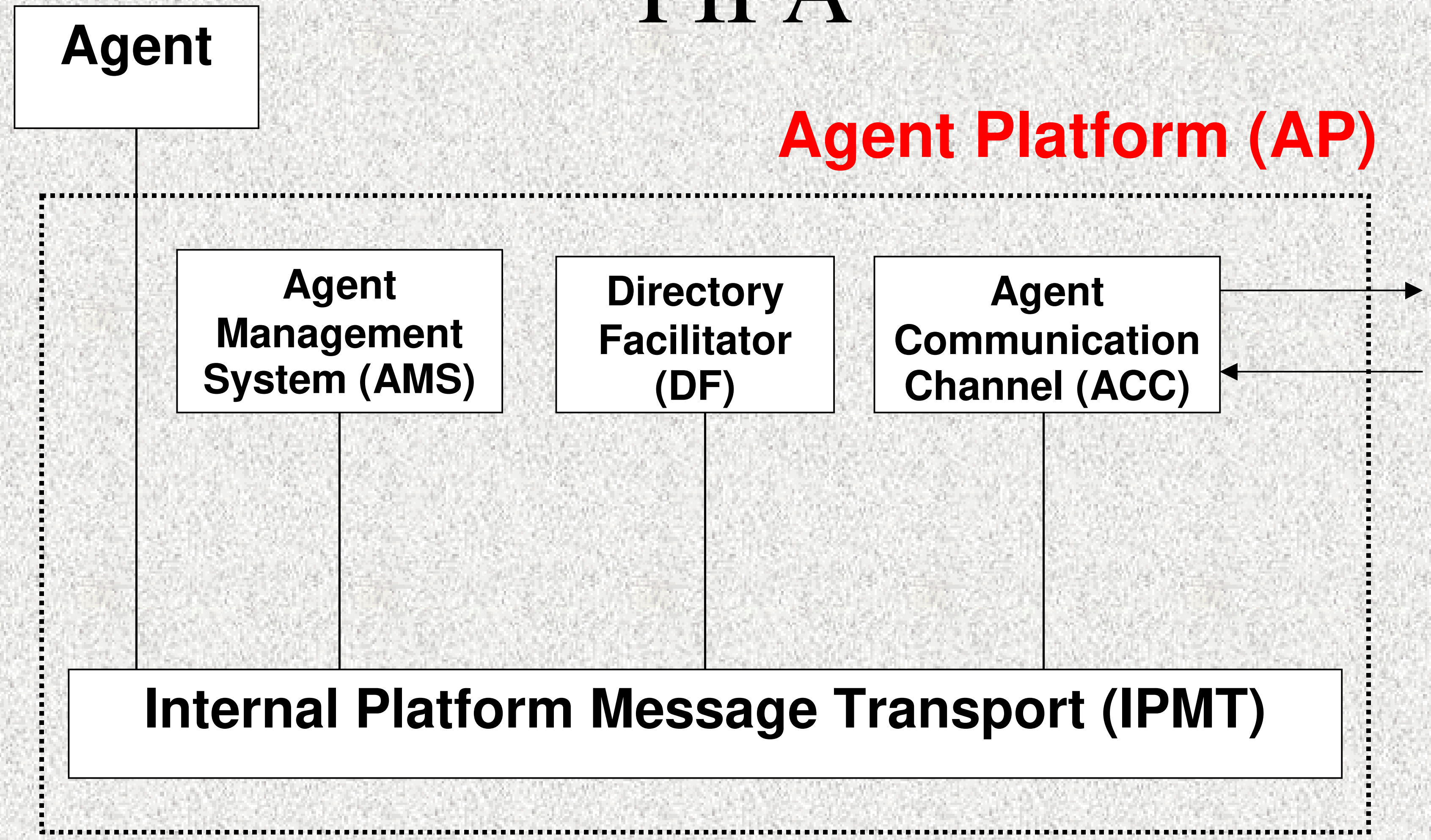
Mandatory specificatios

agent architectures

```
(request
:sender     buyer_agent
:reciever   seller_agent
:content     (buy(apples,10))
:language   prolog
:ontology   market-trader
:protocol   fipa-request
)
```

FIPA Agent Communication Language

FIPA


Agent
Agent Platform (AP)
Agent Management System (AMS)
Directory Facilitator (DF)
Agent Communication Channel (ACC)
Internal Platform Message Transport (IPMT)

Support provided by LEAF

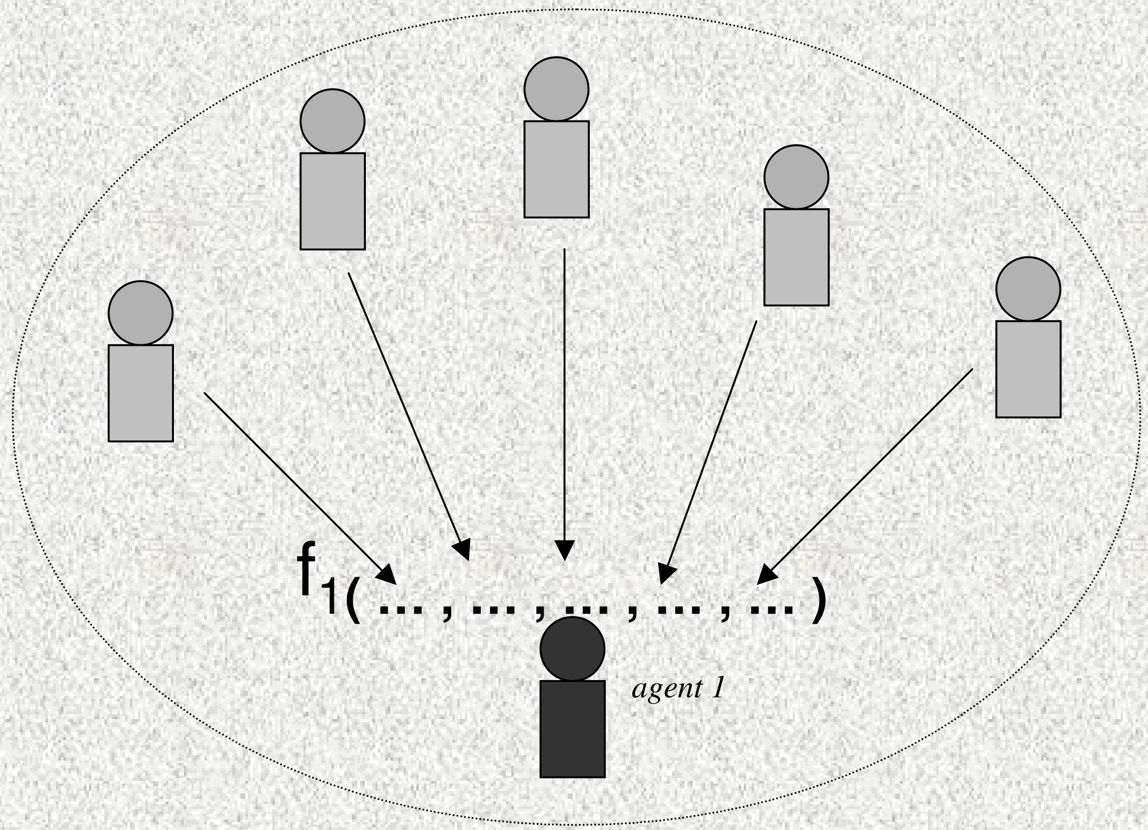
- **Implementation** → Java API, utilisation of FIPA-OS toolkit
- **Coordination** → Utility function assignment
- **Learning** → Mixed learning techniques – Neural Networks, RL etc.
- **Performance Analysis** → Functional utility, Performance utility
- **Scalability** → Multiple utility function assignment

Utility Function Assignment

Utility:

- An analysis tool
- A reward function/coordination mechanism

Utility Function Assignment



Utility Function Assignment

Utility:

- An analysis tool
- A reward function/coordination mechanism

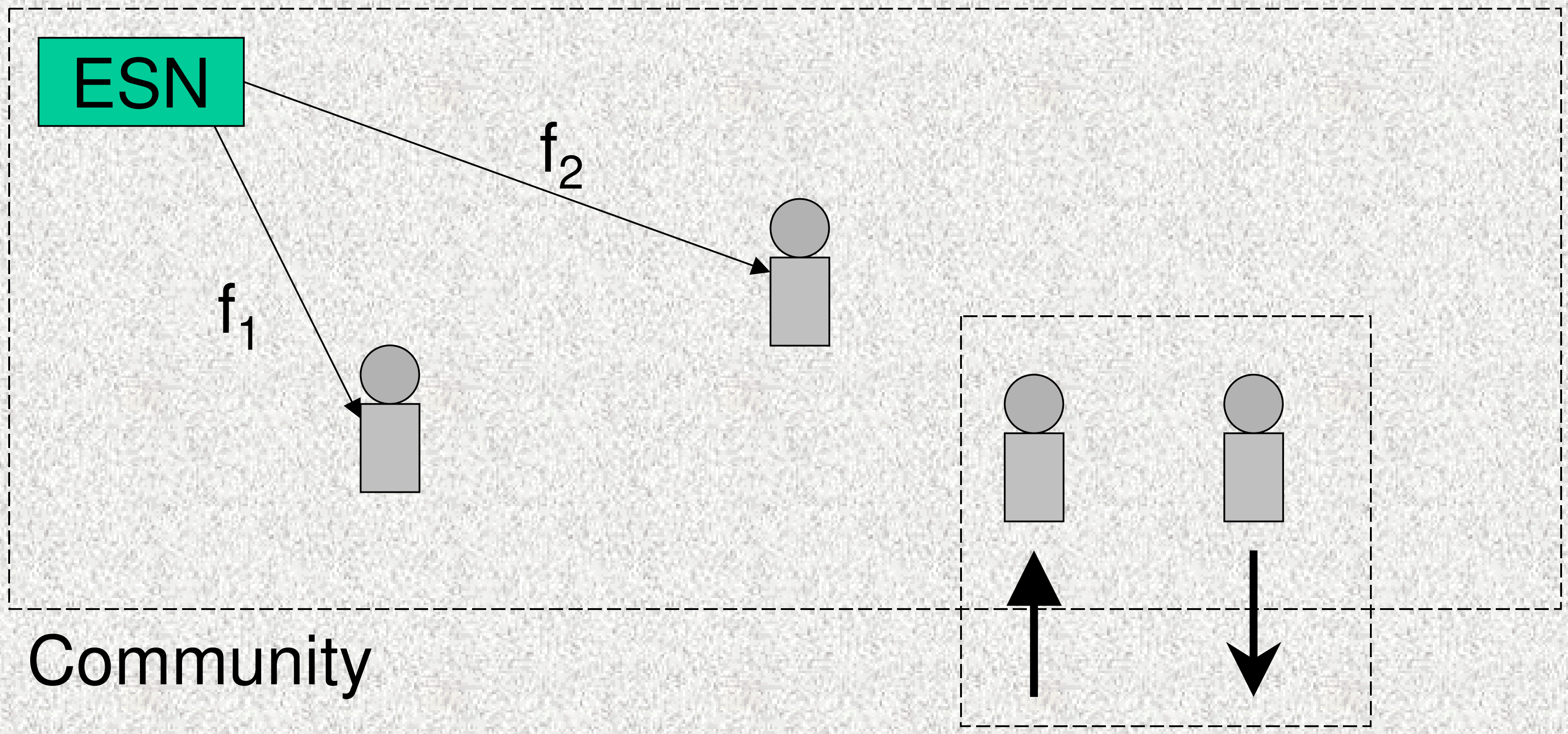
  COIN (Wolpert)
- Agents learn to maximise local utility, which in turn maximises the global utility of a community of LEAF agents
- Performance and functional utility types

Performance Utility

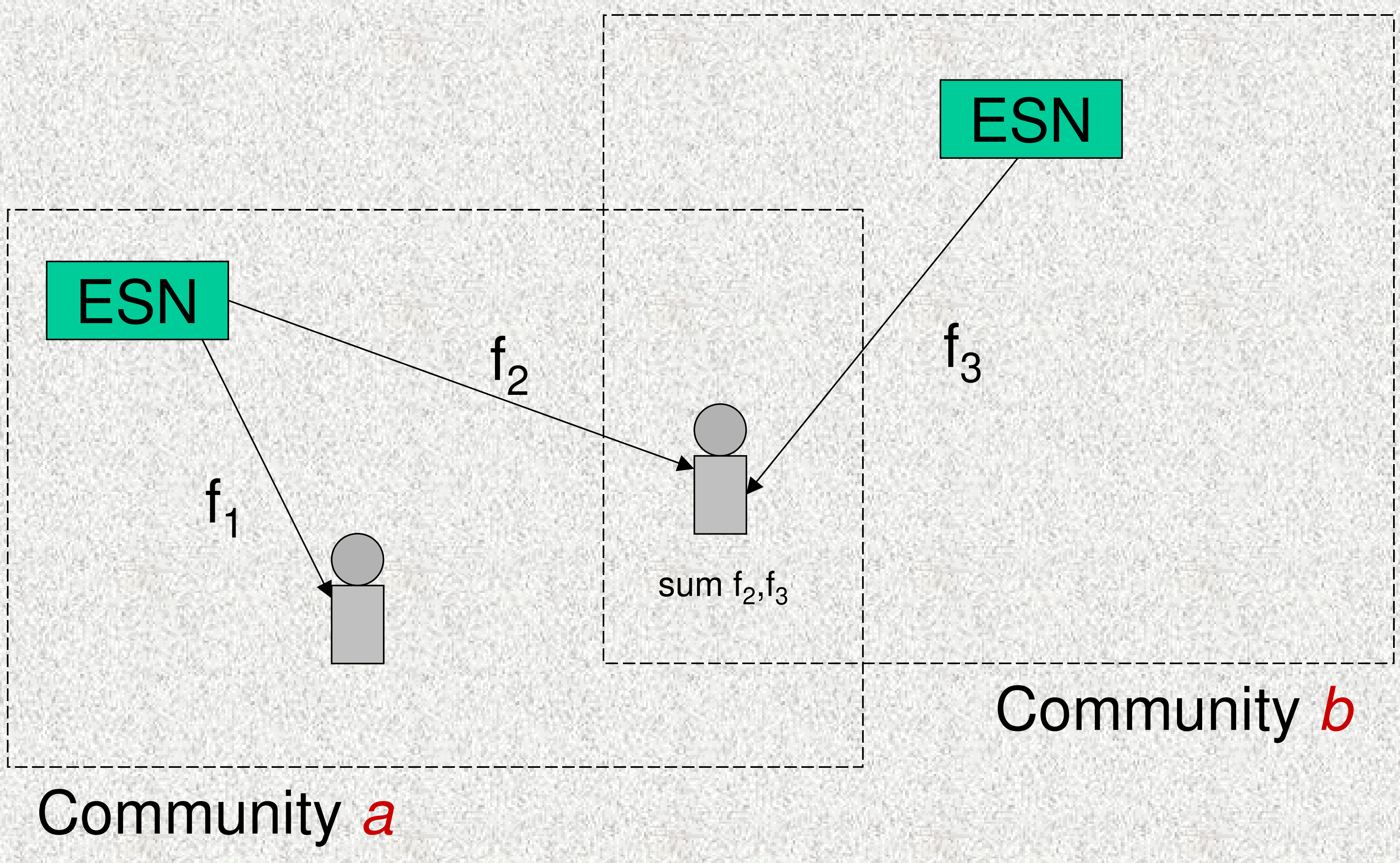
- Provides a utility measure based on performance engineering related aspects.
- The effect of ***implementation decisions*** (algorithms; languages) and ***deployment decisions*** (platforms; networks), can be assessed.
- Performance utility may have a relation to the functional utility of an agent system.

Utility function assignment

- Coordination: utility functions are assigned to agents by an environment service node.

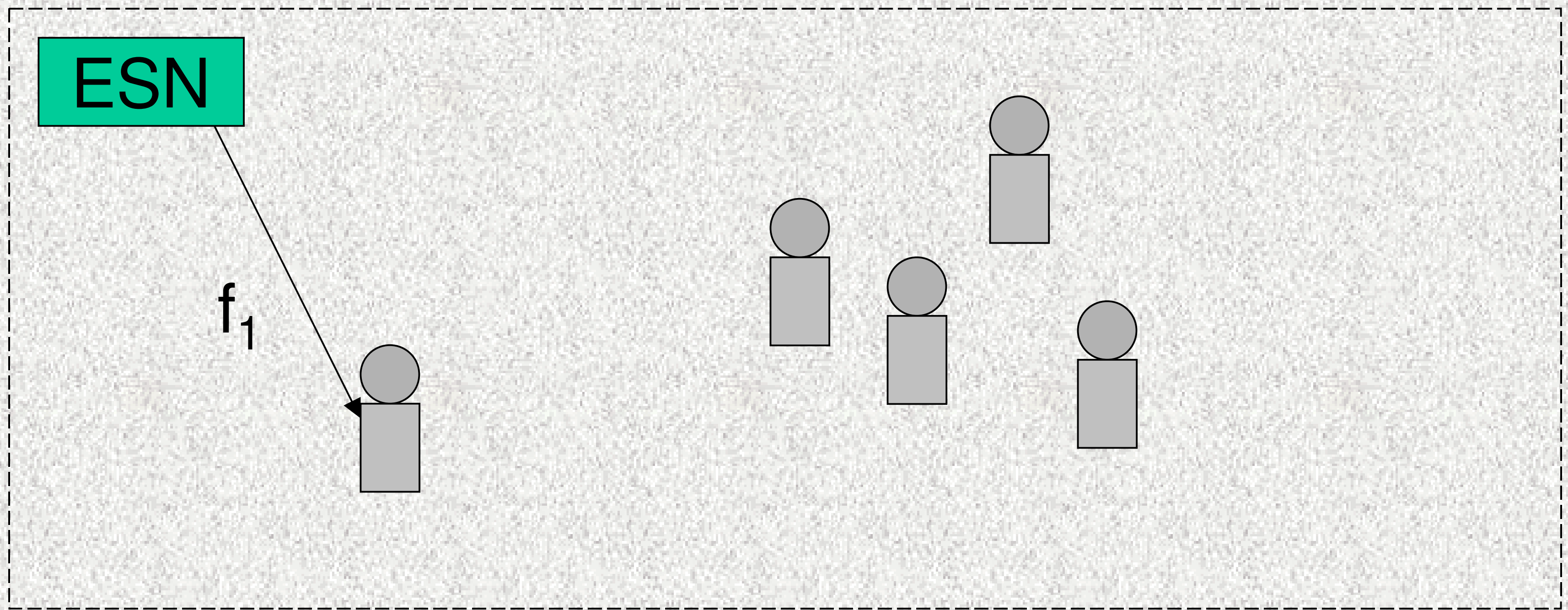


Utility function assignment



Utility function assignment

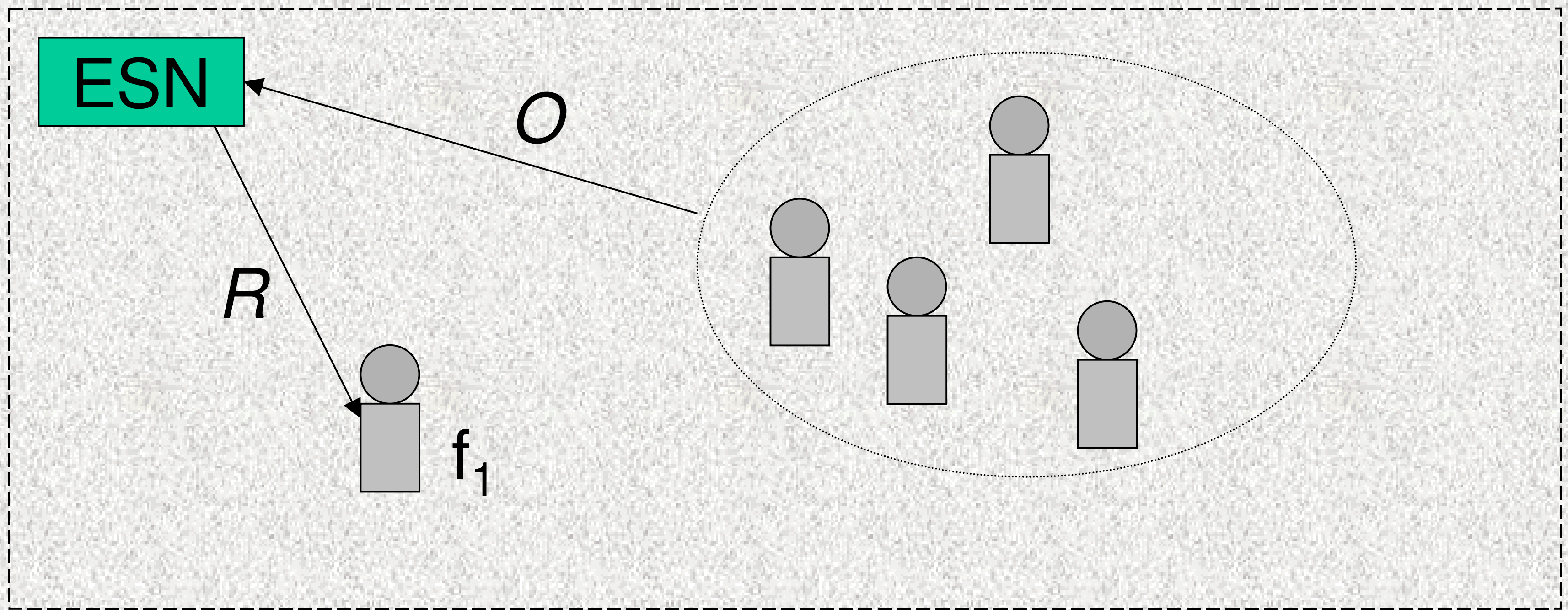
- Utility functions can have parameters not available locally to the agent.



Community

Utility function assignment

- Utility functions can have parameters not available locally to the agent.



Community

O: observable properties
R: remote parameters

LEAF implementation

- LEAF toolkit allows application specific agents/communities to be developed easily
- Toolkit implementation is based on FIPA-Open Source (FIPA-OS) from Emorphia
- Agent behaviours are programmed in Java
- Agents are composed of task components
- Utility functions implemented as Java objects – assigned using object serialization

LEAF agents

- Consist of multiple concurrently executing tasks.
- `JessLeafNode` extends FIPA-OS `JessAgent`
- `LeafNode` extends FIPA-OS `FIPAOSAgent`
- API provided for learning methods – neural network, reinforcement learning

LEAF tasks

- `LeafTask` extends the FIPA-OS `Task` class
- Developers extends the `LeafTask` class
- Task statistics (number of tasks, execution time etc.) used to parameterise performance utility functions
- LEAF automatically maintains task stats – they are available only to assigned utility functions and ESNs.

Utility Function Implementation

- Extend the **`LocalUtilityFunction`** abstract class.
- Implement the **`compute()`** method.
- Functions have access to remote parameters and observable properties.
- Utility functions can be parameterised by observable properties, remote parameters, task statistics and performance data (CPU/memory usage etc.)

Utility Function Implementation

```
package leaf.agentlib.computational;

import leaf.utility.LocalUtilityFunction;

public class ExamplePerformanceFunction extends LocalUtilityFunction {

    public synchronized double compute() {
        return 1 / source.getAverageCompletionTime("JobExecutionTask");
    }

}
```

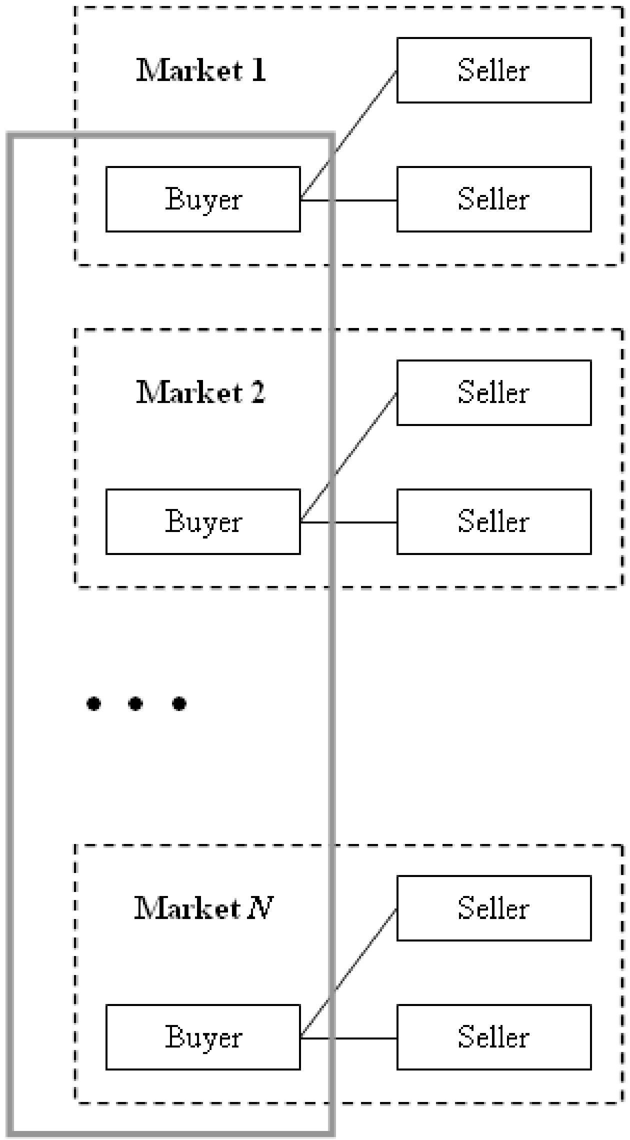
Building LEAF communities

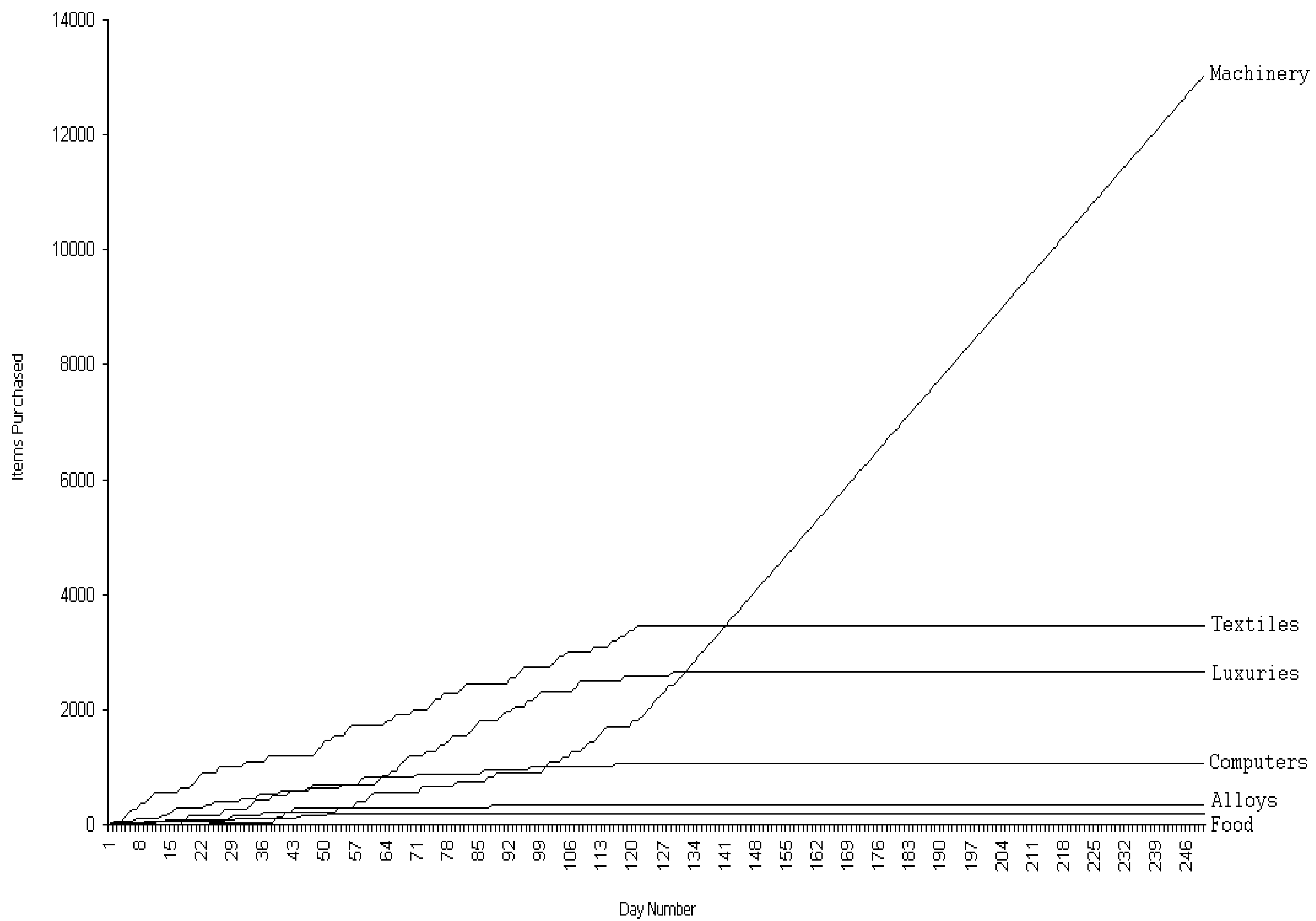
- Agents constructed using *tasks* as building blocks
- LEAF contains a *task library* from which tasks can be selected
- Behaviour is built around the maximisation of local utility (performance & functional)
- ESNs deployed with sets of local utility functions to be assigned to member agents

Applications

Coordination of buyer agents in multiple markets

- A global cost function exists, parameterised by the purchasing decision of the buyer agents
- Buyer agents are assigned utility functions aligned with the global cost function
- Agents use local reinforcement learning algorithms to maximise local utility functions
- Maximisation of local utility functions produces desired overall behaviour





Items Purchased
14000
12000
10000
8000
6000
4000
2000
0
Machinery
Textiles
Luxuries
Computers
Alloys
Food
1 8 15 22 29 36 43 50 57 64 71 78 85 92 99 106 113 120 127 134 141 148 155 162 169 176 183 190 197 204 211 218 225 232 239 246
Day Number

Applications

Management of computational resources

- Agents manage resources and process computational tasks
- Community cost functions are based on the efficiency with which tasks are processed
- Agents may adapt their behaviour – which tasks to accept, execution priorities, resource properties
- Performance utility provides an insight into the benefits of deploying resources of certain platform configurations, and the associated effect on function utility
- The eventual goal is for teams of resource agents to adapt to their environment and function coherently