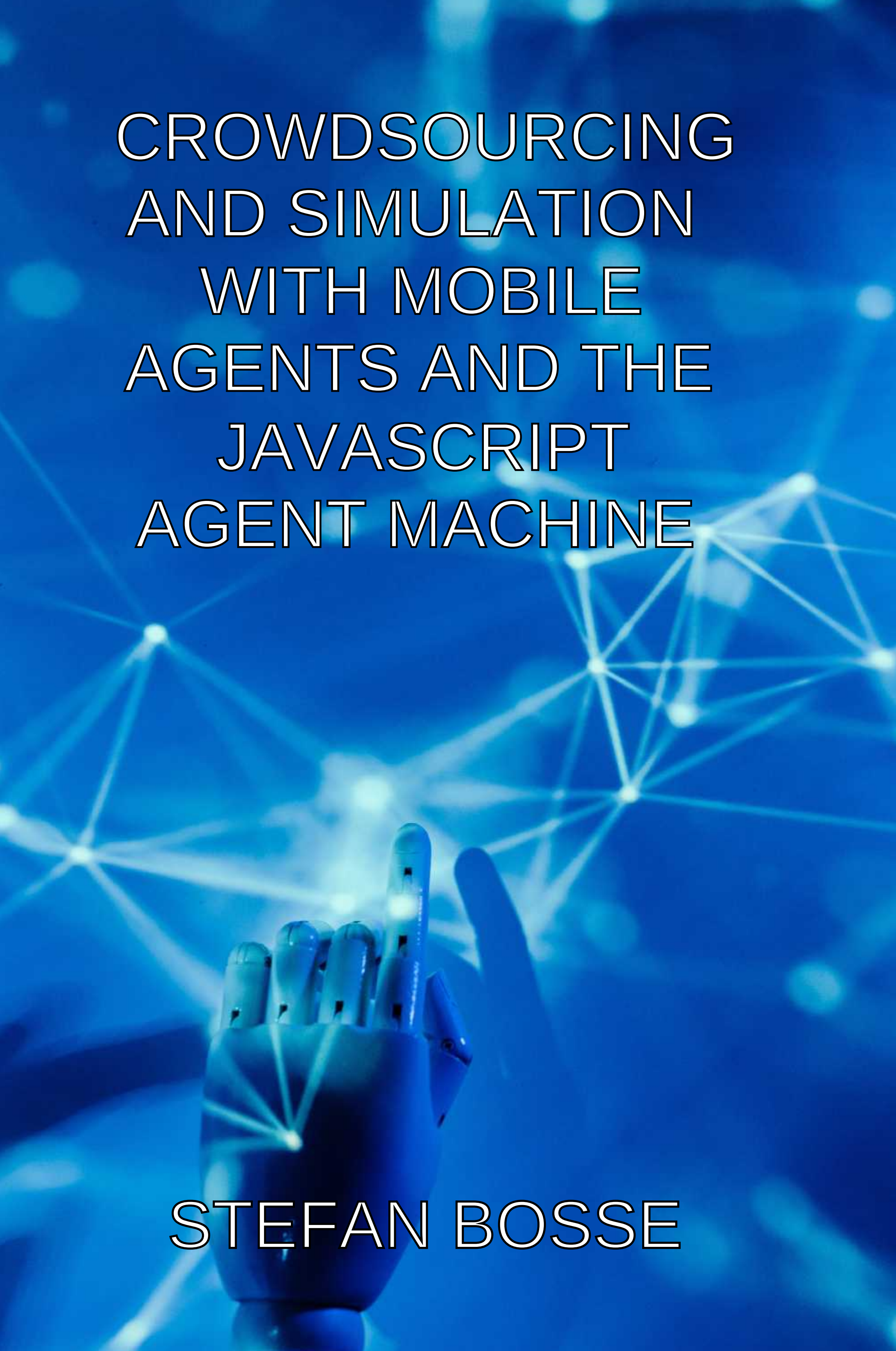

CROWDSOURCING AND SIMULATION WITH MOBILE AGENTS AND THE JAVASCRIPT AGENT MACHINE

STEFAN BOSSE

Crowdsourcing and Simulation with Mobile Agents and the JavaScript Agent Machine

lulu

Leanpub

Crowdsourcing and Simulation with Mobile Agents and the JavaScript Agent Machine

Stefan Bosse

Stefan Bosse
University of Bremen
Department of Mathematics and Computer Science
D-28359 Bremen, Germany
web: www.sblab.de
git: github.com/bslab
orcid: 0000-0002-8774-6141
email: sbosse@uni-bremen.de

Print: Lulu.com

ISBN: 978-1-4710-7813-2

Preface

Using Mobile Multi-Agent Systems, this book tackles the problem of unified and distributed computing in robust heterogeneous contexts, spanning from Internet Clouds to Sensor Networks. The operational gap between low-resource data processing units, such as single microchips embedded in materials, mobile devices, and generic computers including servers, should be closed by a unified agent behaviour model, agent processing platform architecture, and programming framework, supporting real-world deployment as well as simulation. Major features include robustness, scalability, self-organization, reconfiguration, adaptivity, and learning. This book provides a straightforward introduction to creating JavaScript agents using the JavaScript Agent Machine (JAM) requiring only a few lines of code. In a short amount of time, even beginners may develop robust multi-agent systems.

There are countless application areas, including sensor data processing, structural health monitoring, load monitoring of technical structures, distributed computing, distributed databases, and search, automated design, cloud-based manufacturing, mobile crowdsensing (MCS), and surveys. This book has a strong practical focus on MCS. MCS is a useful tool for data mining because it views people as sensors. In addition, agent-based simulation is addressed, finally coupled to real worlds using MCS and digital twin concepts.

With distinct objectives in mind, intelligence and smartness can be defined at various operational and processing levels. One component is the capacity to adapt and be reliable in the face of sensor, communication, node, and network failures without letting the accuracy and integrity of the information computed suffer.

Crowdsourcing and crowdsensing are elaborated in detail after a brief introduction to agent-based notions. If you are solely interested in the agent platform and its programming, you can skip this chapter. The platform is described in connection to the agent interaction and behaviour model. AgentsJS, a subset of JavaScript that is described in depth in a separate chapter, is used to program agents. Pre-compiled libraries and programs are added to the core programming interface, including a simulator. The simulator uses JAM and has the ability to connect to other JAM nodes, allowing for augmented simulation that incorporates the real world.

Finally, an extended example chapter shows various aspects of agent programming with AgentJS and JAM. The software is freely avaiable from https://github.com/bslab/jam. A lot of exercises provide a practical elaboration of agent-based methods, crowdsourcing, and mobile or Web-based surveys.

This book is based on recent scientific work as well as on different lectures I have held at the University of Bremen and the University of Koblenz-Landau. The lectures address the design and deployment of multi-agent systems as well as mobile crowdsensing. Bachelor and master students in computer science, production engineering, and social sciences are the intended audience.

Bremen, August 2022

Stefan Bosse

Contents

1. INTRODUCTION

Agents are well known as human beings that operate on commission for another human being or company to fulfil an order mostly autonomously. Agents are characterized by their social interaction capabilities and social structuring. Autonomy is usually based on planning, cognition, and knowledge.

Agents can be artificial entities, too. They can be used for modelling and simulating natural systems as well as for implementing modern software with superior features. Therefore, agent-based modelling addresses social and natural science on the one hand, and computer science on the other hand.

Agent technology, in contrast, evolved over the past decades and is used to implement or support intelligent systems in unreliable and sometimes unknown environments.

Agent technology is mainly driven by:

- Symbolic Artificial Intelligence,
- Machine Learning,
- Control Theory, and
- Distributed Computing paradigms.

Agent technology is used to master complex dynamic problems in a wide range of fields. Intelligent behaviour is a fundamental feature of agents [1], although the term intelligence is not defined precisely, leading to misconceptions and misunderstanding.

Agents can be used to model and implement complex systems by decomposing the system in simple interacting units, i.e., micro-level modelling, in contrast to macro-level modelling, which uses one big formula to describe the system behaviour. An agent can be a model or a computation paradigm.

This book addresses agent-based computation and agent-based simulation with the JavaScript-based open source software framework *JAM*, available for download on https://github.com/bsLab/jam. Only basic programming skills (in JavaScript) and a Web browser or a command terminal with node.js are required to start exploring the world of mobile agents. Crowdsourcing is a primary application using mobile agents, discussed in detail and with working examples.

1.1 Software Paradigms

There are five major software paradigms reflecting historical changes:

1. Machine programming;
2. Procedural programming;
3. Functional programming;
4. Object-oriented programming;
5. Agent-based goal- and role-oriented programming.

The historical development of programming paradigms is shown in Fig. 1. There are various challenges in the development of modern IT systems:

- Ubiquity $\rightarrow$ 1. Unbound to a location 2. Environmental computing
- Pervasiveness $\rightarrow$ Penetration of computer science into things and devices
- Networking of devices and programs
- Distribution and parallelization of programs
- Intelligence and learning
- Autonomy $\rightarrow$ Without central authorities and control
- Robustness $\rightarrow$ 1. The world changes 2. The world behaves insecure

- Adaptivity → The world has changed
- Delegation of tasks and hierarchies
- Human-machine interface

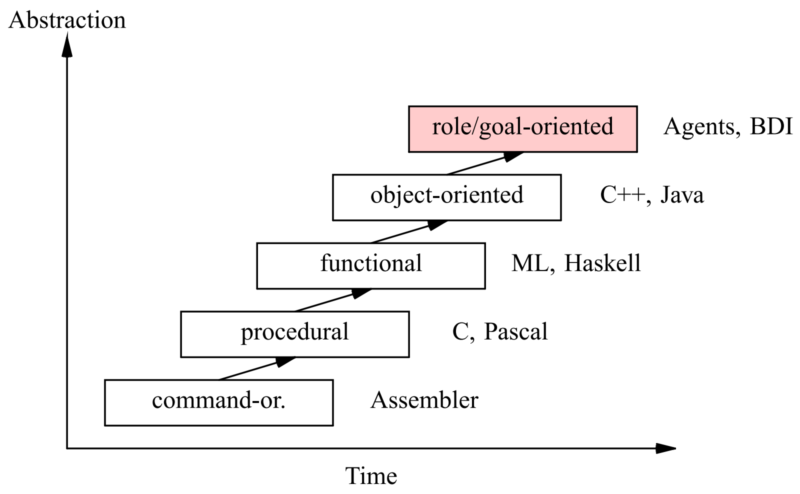

Figure 1. *Historical development of programming paradigms*

1.2 Agents

1.2.1 Autonomy

Agents are autonomous systems interacting in a specific environment (the world). The environment can consist of data and physical entities that can be manipulated by agents. An agent can observe the world through sensors, creating some degree of knowledge about the world. This knowledge can be used to plan decisions and perform actions. Agents communicate with their environment and effect changes in the environment. An agent is always part of the environment, too. Agents are basically characterized by their capabilities and features, discussed in the Section 6.5. There are different understandings and models of agents, basically coupled to diverse scientific communities using agents for different purposes. The science disciplines range from social science to computer science, with an impact from economy, business, and production.

In computer science, the agent concept is used to establish a paradigm shift from the traditional and still widely used client-server architecture towards distributed agent systems performing data processing and communication, as illustrated in Fig. 2.

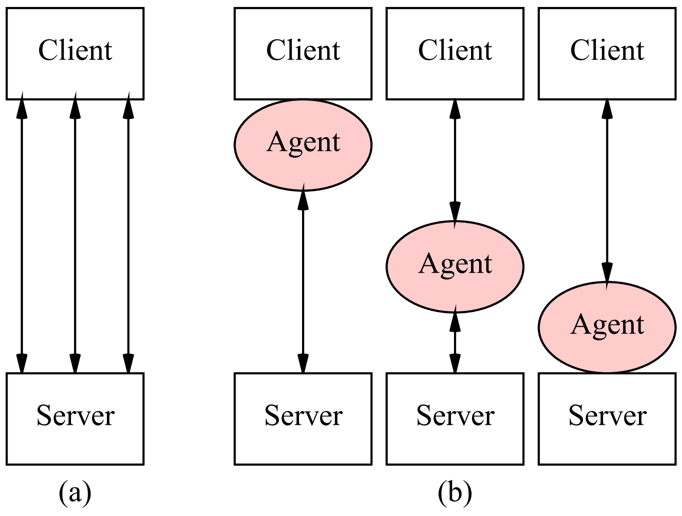

Figure 2. *From the client-server communication architecture (a) towards distributed agent-based and agent-controlled communication (b)*

In addition to the general agent concept, mobile agents are defined by two concepts: Mobility and agency [2]. Mobile agents can reduce communication costs and can increase robustness through adaptivity and replication.

1.2.2 Metrics

Agents can be classified by the following metrics:

1. Physical Agents
    i. Hardware Agents
    ii. Software Agents
    iii. Social Agents
2. Computational Agents
    i. Interface Agents (Human-Machine and Machine-Machine)
    ii. Data Processing Agents
3. Mobile Agents
    i. Physical Agents
    ii. Computational Agents

In this book, two different agent classes are used in conjunction:

Physical Behavioural Agents
represent physical entities, i.e., individual artificial humans, animals, vehicles, or more general machines. This agent category is divided into hardware agents, i.e., being robots or machines, and software agents, representing physical entities. A sub-set of software agents are social agents, which model the behaviour and interaction of social systems. Agents in simulation are considered as physical agents, too, since they represent physical entities.

Computational Agents
represent software and processes performing computational tasks, i.e., used for distributed data processing and digital communication, or implementing chatbots. Computational agents require an Agent Processing Platform (APP) for their execution.

Mobile Agents

can be considered as a super-set of physical and computational agents. Mobility of physical agents means the capability of spatial movement of the hardware or software agents (or the migration of social entities). Mobility of computational agents, represented by computational processes, means the transfer of a process snapshot between agent processing platforms.

Among the basic agent classifications from above, dividing the agents by their impact on and interaction with the environment, agent-based methods can be classified by their deployment and usage with five domains:

1. Agent-based Modelling (ABM)
2. Agent-based Simulation (ABS)
3. Agent-based Modelling and Simulation (ABMS)
4. Agent-based Computing (ABC)
5. Agent-based Simulation and Computing (ABX)

This book discusses the different modelling and deployment fields of agents and finally presents a fusion of all four levels (ABX) using the JavaScript Agent Machine.

1.2.3 Model

An agent, independent of its classification, interacts with an environment by basically two mechanisms:

1. Perception: performing environmental sensing and interpreting sensor values;
2. Action: performing modification of the environment.

The simplest agent behaviour model and architecture is shown in Fig. 3 and consists of a cyclic data flow between the agent and the environment.

An agent poses a set of sensors $S=(s_{1,}s_{2,}..,s_n)$ used for perception, and a set of actuators $A=(a_1,..,a_m)$ used for the modification of the environment.

The basic black box model of the agent's behaviour is a function *Ag* that maps the sensor input vector *S* to the actuator output vector *A*:

$$Ag\left(\vec{S}\right) : \vec{S} \to \vec{A} \tag{1}$$

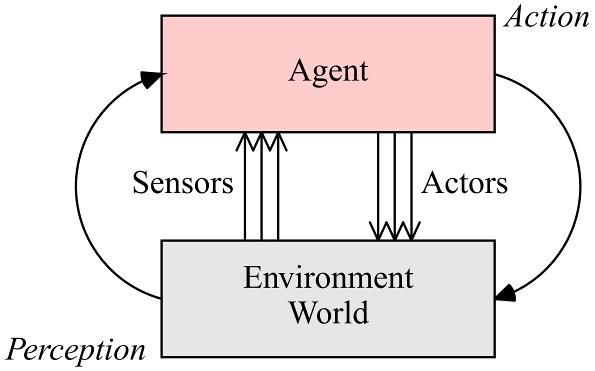

Figure 3. *Simple reactive agent behaviour model and architecture*

Sensors can be classified in:

1. Physical Sensors - intrinsic and extrinsic measuring different physical properties:
    - Temperature (ambient, body, remote, device, component)
    - Motion (Acceleration, rotation, translation)
    - Position (Global, local, absolute, relative)
    - Light (Intensity, colour, wavelength, spectrum)

- Radiation (electromagnetic, nuclear)
- Pressure (Air, loading)
- Humidity, Moisture

2. Data-driven logical sensors delivering data from:
   - WEB Content
   - Data bases
   - Networks
   - Social Media
   - Text and Graphics
3. Virtual Sensors (Aggregated from physical and logical sensors)

Actuators can be classified in:

1. Actuators that change the physical world
2. Logical Actuators (Data) modify the digital world

1.3 Modelling Concepts

There are basically two different modelling approaches for systems:

1. Top-down Modelling;
2. Bottom-up Modelling.

Top-down models start with the specification of the global system level and divide the system model into smaller modules directly derived from and coupled to the system level specification. The behaviour of the system level is known in advance and can be verified. The behaviour of the modules is directly derived from the system level. In contrast to top-down modelling, bottom-up models start with a set of simple interacting cells, eventually composing a system from these cells. The system level is a result of the interaction of cells and their behaviour, not known and specified in advance. Agent-based modelling belongs to the bottom-up class.

1.4 Capabilities of Agents

An agent provides a set of features and capabilities that distinguish agents from traditional software:

- Autonomy
- Self-* capabilities:
  - Self-organization
  - Self-coordination
  - Self-adaptivity
  - Self-optimisation
  - Self-learning
  - Self-awareness
- Divide-and-Conquer enables the composition of complex systems with simple entities similar to cellular automata
- Resilience and robustness
- Learning and adaptivity of behaviour
- Planning and Cognition
- Knowledge Representation, Induction, and Deduction
- Symbolic Reasoning
- Social capabilities like
  - Coordination
  - Cooperation
  - Negotiation

- Emergence behaviour in multi-agent systems, i.e., only the aggregate behaviour of the system is relevant, not individual behaviour of agents (and failure)
- Loosely coupling between agents and agents and the execution platform (API)

Agents are similar to cellular automata, which are used to model complex systems by connecting simple basic cells. The main difference between cellular automata (usually only implementing simple input-output functions) and agents is their interaction distance, which is short range (limited to the neighbourhood) in cellular automata and arbitrary in agent-based systems.

1.4.1 Emergence

The observation of emergent properties is the global or collective behaviour of a system composed of a collection of loosely coupled simple entities or agents.Emergence is a result of self-organization, self-adaptivity, and self-coordination. On one hand, the emergent behaviour vanishes if the system is broken down into individual parts (operating in isolation) or if a relevant fraction of individual parts of the system is removed. On the other hand, the removal or malfunction of a small fraction of individual parts does not affect the emergent properties of the global system significantly [3].

The emergent behaviour of the system can neither be modelled at an individual entity level directly, nor at a system level by a global function. This emergent behaviour is usually observed. Although, the emergent behaviour is determined by the individual behaviour of the agents. But multi-agent systems aim to reverse engineer emergent phenomena. Typical examples are ant colonies, the economy and trading, and the immune system.

An example of emergent behaviour is segregation, i.e., long-term movements of individuals forming spatial clusters, e.g., in town areas. The Shilling model is a well-known clustering model based on very simple behaviour and interaction. The Sakoda model is less known, but can be parametrized at an individual level. Some simulation results of globally visible segregation effects are shown in Fig. 4. The Sakoda model is based on an individual local utility (satisfaction) function that calculates a utility to stay at the current or at another place under specific environmental conditions (other agents in the neighbourhood) [4].

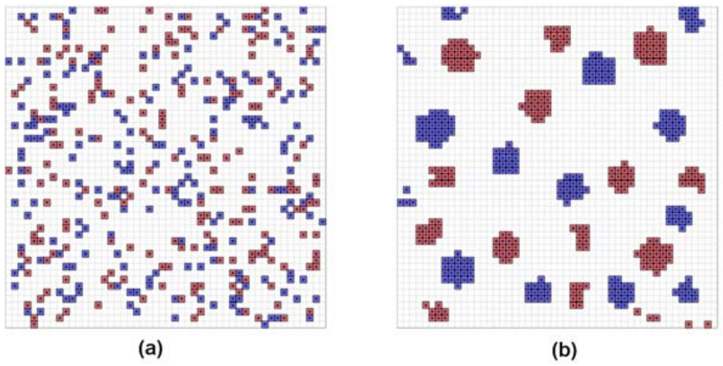

Figure 4. *(a) Simulation world with 200/200 randomly distributed artificial Sakoda behaviour agents of class a/b (blue/red squares) (b) Simulation world with social organisation based on mobility, forming strong isolated homogeneous clusters after 200 simulation steps [4]*

1.5 JavaScript

The agent processing platform as well as the programming of agents is done with pure JavaScript. JavaScript, originally named ECMAScript, is a partially functional and object-oriented imperative programming language with one control flow (strict single-threaded without pre-emption) and automatic memory management. JavaScript programs consist of polymorphic variables and functions. JavaScript is dynamically typed, i.e., data types and function interfaces are evaluated at run-time. There are basically only six defined core data types:

1. Number (commonly double precision float)

2. String (immutable)
3. Boolean
4. Object
5. Function
6. Undefined

Arrays and structures are both objects. Arrays are poly-sorted, i.e., different elements of an array can be of different data types. Arrays are commonly created incrementally by the *push* operation. Array elements are accessed by the bracket operator. Structure elements (object properties) are accessed by the dot operator by providing the property name as an identified. Alternatively, the property can be selected by the bracket operator, too, by providing the name as a string. Pure data objects (records) and arrays can be created on the fly:

```
var a = []
for(var i=0;i<10;i++) a.push(i);
a[1]=a[2]-1;
var img = {
  x:1,
  y:-1
}
var absimg=Math.pow(img.x,2)+Math.pow(img.y,2)
var struct1 = { x:1, y:2 }
struct1.x=struct1.y
struct["y"]++;
var array1 = [1,2,3,'A','B']
array1[0]=array[1]+1
function foo () { .. }
var foo2 = foo
var foo3 = function () { .. }
var foo4 = () => { }
```

Object classes were originally supported by adding prototype functions to object constructor functions:

```
function cls() { this.data = ε; .. }
cls.prototype.method = function () { this.data= ε }
var obj = new cls()
obj.method(..)
```

Each prototype function can access the instatiated object by using the `this` variable. All objects instantiated from the same constructor function share the same set of prototype functions.

A variable can hold any value, but initially the value `undefined`.

Any value can be assigned at anytime to a variable, including named and anonymous functions. Functions can be serialized anytime to text again, an important feature for object (agent) snapshots and checkpointing. All non-cyclic and objects without prototype bindings can be serialized to text in the JSON format.

There are only two scopes of identifiers (i.e., variables and functions): The global scope and the function body scope. All variable definitions in the same scope are merged, function definitions are overridden in the order they appear:

```
var top=1
function foo (localparam) {
  var local=2
}
var top=2 // overrides line 1
```

Function are objects, too. Each function provides at least the *prototype* property referencing the set of dynamic functions, and can be extended with user defined properties, e.g., static functions or data values:

```
function foo () {}
foo.name = 'foo'
foo.info = function () { return foo.name }
```

Text serialization and deserialization of data and functions are major features of JavaScript important for the support of mobile agents via process snapshots.

2. CROWDSOURCING

At the beginning of this chapter, a short terminology should be introduced that will be used throughout the book:

Sensor
Physical converters that convert one physical quantity into another physical quantity (usually analogue, electrical or digital). Examples of physical quantities are temperature, light intensity, acoustic waves, and mechanical strain.

Virtual Sensor
Functions with an input and output interface for further processing and fusion of sensor signals. Aggregate variables (e.g., average) can be computed by virtual sensors.

Sourcing
Creation of added value: Source of information with the provision of data but also the provision of work performance; Collection of goods;

Sensing
Process of acquisition and collection of sensor data $\boldsymbol{X}=\{x_1,..,x_i\}$, typically periodically or event-based along the longitudinal time axis, i.e., $x=x(t)$.

Perception
Perception and interpretation of sensory information