**LightJason, a Highly Scalable and Concurrent Agent Framework: Overview and Application**

Demonstration

Malte Aschermann, Sophie Dennisen, Philipp Kraus, Jörg P. Müller

Technische Universität Clausthal, Institut für Informatik
Clausthal-Zellerfeld, Germany, 38678

[malte|philipp|sophie]@lightjason.org
jmue@tu-clausthal.de

**ABSTRACT**

Multiagent systems (MAS) provide useful abstractions for modelling and simulating complex socio-technical systems. However, existing open-source platforms suffer from serious issues including limited scalability, inflexible software architecture, and poor support for web deployment. This demo presents an overview and an application of LightJason, a highly scalable Java-based platform for agent-oriented programming (AOP) and simulation. We outline the architectural features of LightJason and showcase its applicability using an example of a browser-based web application implementing a traffic serious game devised to teach an interdisciplinary student team in MAS and AOP.

Demonstration video: https://vimeo.com/lightjason/aamas2018

**KEYWORDS**

Agent-oriented programming and simulation; BDI modelling; agent platform; traffic simulation; scalability

**ACM Reference Format:**


**1 INTRODUCTION**

The modelling, implementation, and simulation of socio-technical systems (STS) [14] is a major challenge for computer science. STS are systems such as Smart Cities in which heterogeneous human and automated entities are embedded in large-scale dynamic environments, form organisations, and act and interact in complex ways. Modelling STS requires appropriate models and mechanisms, such as micro-meso-macro architecture [12], normative control, computational mechanism design [13] and social choice [7], or game-theoretic models. From a computing perspective, dealing with STS imposes hard requirements on underlying platforms and software frameworks in terms of software quality, scalability, flexibility of software architecture, the ability to integrate with third-party software, and to support web deployment and execution on cloud/high-performance computing (HPC) platforms. MAS provide useful abstractions for modelling and simulating complex STS. However, existing open-source platforms like Jason [6, 16], GOAL [8] or Jade [5] fall short of satisfying the abovementioned requirements: Flaws in algorithmic design slow down performance; inflexible software architectures (e.g., built-in proprietary runtime systems) make integration with other systems (e.g., traffic simulators [4, 9]) hard, and are mostly not geared to support web deployment and web-based visualisation of application, and to use state-of-the-art HPC/cloud platforms. For our findings on algorithmic and architectural comparison, we refer to [1–3]. In this demo, we present an overview and an application of LightJason, a concurrent Belief-Desire-Intention (BDI) multi-agent framework for creating multi-agent systems with Java. LightJason has been inspired by Jason and AgentSpeak(L) [11]; however, it is based on a newly developed software architecture and, compared to Jason, proposes a new concurrent semantics of the agent cycle and a very efficient implementation of agent perception. Furthermore, the logic programming language AgentSpeak(L++) used in LightJason provides considerable extensions to AgentSpeak(L), e.g., lambda-expressions, multi-plan and -rule definition, explicit repair actions, multi-variable assignments, parallel execution and thread-safe variables. LightJason offers a modular runtime system. By using object-oriented concepts and Java generics, tailored runtime instances can be created for an application, and LightJason can be easily integrated into third-party applications.

For further background/technical information and practical examples for multi-agent applications of LightJason we refer to [3] and the documentation at http://lightjason.org.

**2 SPEEDING GAME APPLICATION**

A LightJason-based browser application for teaching MAS. In this demo, we focus on how LightJason supports integrating agent technology into state-of-the-art browser-based web applications, which is difficult to handle with existing agent frameworks, which are mostly stand-alone with “hard-wired” integrated development environments (IDEs), runtime (RT), graphical user-interface (GUI) and source editor. In LightJason, these components are modular, which makes the framework lightweight, adaptable and easy to integrate.

Scenario. Our use-case model describes a highway traffic scenario which is used to teach students multiagent system (MAS) and AOP. The actors (vehicles, road segments with speed limits, traffic rule enforcement) are modelled as BDI agents. For details on how we modelled the scenario via ASL++ scripts, see https://git.io/vxh0g. In this demo, students have the task to write an agent program to...
steer a specific vehicle through a fixed-length road (four lanes with two directions) with changing speed limits and a black-box traffic rule enforcement (e.g., speed cameras that will be activated using some enforcement strategy unknown to the students).

Vehicle agents can perceive information about segments, current speed, allowed speed and an estimation of expected penalties. If and when penalties are imposed is also modelled by means of agents, governing areas of varying traffic limits and the environment. A well-performing vehicle driving strategy should strive after the user optimum, where the utility function reflects the goals to minimise travel time and penalty for speeding, while avoiding collisions with other vehicles.

**Technical realisation.** We chose a browser-based application (WebUI) with a web server for the simulation backend and an interface for visualisation. In LightJason BDI logic is programmed as scripts using a code editor, and translated into Java objects which are executed by the RT. For visualising the current state of the environment and agents, we used the Phaser1 game engine. Fig. 1 illustrates the architecture of the resulting web application. The *Speeding Game* application consists of a JAR-bundle of programs with all necessary components and a web-based GUI for modelling the agents and accessing lecture material (see Fig. 1).2 The RT executes the agents concurrently and in a scalable manner. To give feedback to the users regarding the performance of their strategies, penalty charts (Fig. 2b) are shown via Chart.js3. Ajax and WebSockets are used for the communication between browser frontend and Java backend. The web server component uses Jetty with a Servlet and WebSocket endpoint for the web server and Bootstrap for the browser interface.

### Figure 1: Architecture of the *Speeding Game* application

![Architecture of the Speeding Game application](https://example.com/figure1.png)

#### 3 EVALUATION AND CONCLUSIONS

A first evaluation of the *Speeding Game* application was carried out in a small-scale teaching study. In summer 2017, the Ph.D. students of the interdisciplinary Research Training Group SocialCars4 attended a series of two-day simulation labs as part of their initial qualification programme. The students’ background was very interdisciplinary, covering the areas of psychology, informatics, traffic engineering, operations research, geoinformatics, and communication technology. The purpose of the simulation labs was to increase the mutual understanding of research fields. Each lab contained a mix of theoretical lectures and practical hands-on workshops. In this context, we had the opportunity to present and evaluate the *Speeding Game* application in an AOP lab for the abovementioned scenario. The objectives of the lab were twofold: (i) Get an overview of MAS with a practical introduction to MAS models and frameworks, while 2) taking various degrees of initial knowledge and skills regarding programming in general and AOP in particular into account. The main goal for the participants was to learn AOP for traffic simulations by completing three main assignments: (i) Get familiar with the LightJason programming language AgentSpeak(L++); (ii) Understand the execution behaviour of agents by analysing and trying out a given baseline strategy based on an extended Nagel-Schreckenberg model [10]; (iii) Create a new and competitive driving strategy. At the end of the lab, students submitted their results, the best strategies were determined in a contest, and prizes were awarded. The didactic concept of the lab followed the flipped-classroom (FC) [15] method, such that the PhD students could put their theoretical knowledge of accompanying lectures into practice. Lectures and exercises alternated over the two-day workshop. While completing their assignments, students were us-

#### Figure 2: Excerpts of simulation web interface

![Excerpts of simulation web interface](https://example.com/figure2.png)

1https://phaser.io/
2For details and source code, see https://git.io/vATPU and https://git.io/vATPq.
3http://www.chartjs.org/
4https://socialcars.org/

See https://goo.gl/forms/ftnnmmMrOV3g2QOFE2

Acknowledgements. We thank Ehsan Tatasadi for creating the web interface.
REFERENCES


