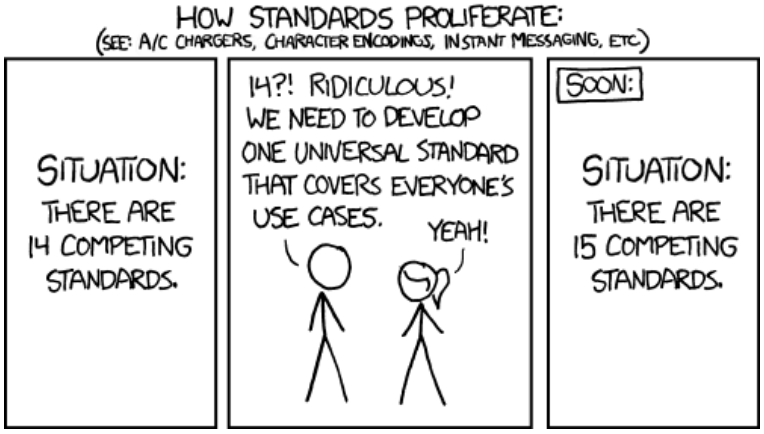
SCALABM

DAVID R. PUGH, DANIEL F. TANG, J. DOYNE FARMER

1. Objective...

Our objective is to create a *user-friendly* toolkit for building *scalable*, *data-driven* agent-based models (ABMs) of *economic* systems.

1.1. **Motivation.** Currently the economic ABM community lacks a coherent set of standard tools for testing, developing, and validating their models. The lack of such a toolkit has significantly hindered the progress of agent-based modeling.[1] What we have outlined in the previous section is our vision of what such a "standard" should look like. However, in order to avoid...



...we need to maximize the chances that our toolkit gets adopted as a standard. There are a number of facets to our vision that we believe increase the likelihood that our toolkit could be adopted as a standard...

- Network effects are key to establishing a standard. We are leveraging recent hardware and software advances (i.e., AWS, Heroku, Scala, Akka, Spark, Cassandra, Neo4j, etc) that are all open-source, free even for commercial-use (specifically Apache 2.0), and that have already generated substantial networks effects in private industry. Users (i.e., undergraduate and graduate students, post-docs, etc)

Date: September 17, 2015.

[1] TODO: Compare and contrast our approach with EURACE (and its derivatives), JMAB, Jas-mine, CRISIS, Repast, Swarm, etc. In particular focus on applications that are economics focused.

who adopt our toolkit will be learning to tools that will substantially increase the probability of finding a high-paying job (if they choose to leave academia). Not clear that learning FLAME, Repast, Swarm, Netlogo, Mason, etc would be as useful outside academia.
- Credibility is key to establishing a standard. If you were starting a new scientific ABM project, would you be willing to use software if the source code wasn't available, documentation was sparse, and no unit tests had been written? We are following best practices for software engineering that have been used in previous, successful, open source projects and that have found widespread adoption in private industry. In particular, we will place our source code under version control, develop a rigorous set of unit tests, and strive to maintain a high degree of test and documentation coverage, throughout the lifetime of the project.

...additionally...

- We should strive to release (early and often!) models from the existing ABM literature as simple example use-cases.
- We should actively seek out partners in private industry for software development expertise. Many (all?) previous efforts to design toolkits for agent-based modeling have failed due to insufficient attention to software engineering issues. We have been developing networks and contacts in private industry (i.e., Typesafe, Datastax, etc.) that can help us with some of the software engineering challenges.
- We should actively seek out partners in private industry for funding of specific projects that would represent use-cases for our toolkit. The likelihood of finding partners in private industry should be increased by our use of a permissive, open-source license.

1.2. **Requirements.**

1.2.1. *User-friendly...* We expect that users of our ABM toolkit can be classified as either consumers or producers. Consumers are users whose primary objective is to learn about the mechanisms driving a model's key results by playing with model parameters and components. Producers are users whose primary objective is to develop new models using some combination of existing and novel model components.

We want consumers of models built using our framework, particularly those not directly involved with development of any particular model, to be able to easily interact with a model in order to develop intuition and understanding about the mechanisms driving that model's key results.

- Models built using our toolkit should have browser-based user interfaces (UIs). Implementing browser-based UIs would allow us to leverage existing, high-quality Javascript libraries for real-time data visualization.

- Users should be able to "play-with" the model (i.e., tweak model parameters, substitute model components, etc.) from the browser in real-time (i.e., whilst a model is running).

We also want to develop a framework that minimizes developer time when either building a new model (or re-configuring an existing model).

- Models built using our toolkit should be composed of mostly existing components. This reduces development time for a new model to that needed to create a few novel components together with the time needed to wire all the desired model components together.
- The process of wiring model components together, sometimes called dependency injection (DI), should be as simple and transparent as possible.[2]
- All model configuration should be done in a *single* file (i.e., some type of application configuration file). Using this application configuration file it should be possible to reproduce a run of a particular model.

1.2.2. *Scalable...* Aggregate behavior of many (most?) real world social systems fundamentally depends on system size. Therefore in order to accurately model system dynamics we need to be able to run our models as close to observed scale as possible. We need to start thinking about building models "as if" we were developing web applications so that we can leverage recent hardware (i.e., massively multi-core servers, cloud computing architectures such as AWS, Heroku, and Mesosphere, etc) and software advances (specifically Scala, Akka, etc) that are driving the proliferation of large-scale, distributed web applications in private industry.

1.2.3. *Data-driven...* We want to build models that can be validated against empirical data and therefore we need to design our toolkit to facilitate this. Again we should think of our models "as if" they are web applications. Web applications consume streams of input data and produce streams of output data; some output data streams are processed and then used again as new input data streams. With all of this data flowing around, database integration and data processing are crucial components of web application *design*. We should design our toolkit to integrate cleanly with database architectures, such as Apache Cassandra and Neo4j, and data processing engines, such as Apache Spark, that are specifically designed to support large-scale web applications.

1.2.4. *Economic...* Finally, we are not seeking to build a general purpose toolkit for agent-based modeling. Rather we want to build a toolkit that is designed to facilitate the construction of agent-based economic models. An economy is populated with many seemingly disparate types of agents (i.e.,

[2] There are many popular DI approaches (i.e., Guice, Spring, etc). While our framework should not depend on any particular DI library, we should think carefully about what DI library we choose as I expect many users will just mimic our choice.

consumers, producers, financiers, government, some markets, etc). Our goal is to distill the core essence (in terms of data and behaviors) of these different agents into a multi-layered Application Programming Interface (API) defining a generic *economic agent* that can then be specialized to the various types of economic agents needed for any particular model. Our hope is that by doing this we can reduce the effort needed for the parts of agent-based modeling that consume a great deal of software development time, such as accounting or contract enforcement. We also hope that by identifying the key components we can build standard, highly modular interfaces that make it easy to interchange components of models.

1.2.5. *Reproducible...* Results of many (most?) ABMs are not easily reproducible. More often then not, source code for models used in published articles is not publicly available. Reproducibility is further hindered by the lack of use of "best practices" for software development (in particular unit testing). Documentation is typically lacking.

In order to address these issues and make models built using our toolkit reproducible we should develop the toolkit *from the beginning* as an open-source project and should exemplify "best practices" in software engineering: version control using Git and GitHub; continuous integration of unit tests using Travis CI or Jenkins; static analysis of source code using Codacy, Code Climate, etc; documentation should be extensive, written along with the code, and publicly hosted on GitHub.

1.3. **Coarse-grain architecture.** We want to mimic the layered architecture of a modern distributed web application. The coarse-grained archictecure of ScalABM consists of five layers.

- Cluster Management Layer: Base layer for the entire framework. We will not need to build this layer. We can simply use third-party cluster management tools. Mesos, Amazon EC2, Hadoop/Yarn. Probably best to package our application up in a "container" of some kind (i.e., Docker, etc) and then ship it to a third party cloud computing service that manages a cluster and who can then scale up our application. Note that, unlike EURACE, we will NOT need to have access to a university or governmental super computer to scale up our application.
- User Interface Layer:
- Data Analytics Layer:
- Model Layer:
- Database Layer:

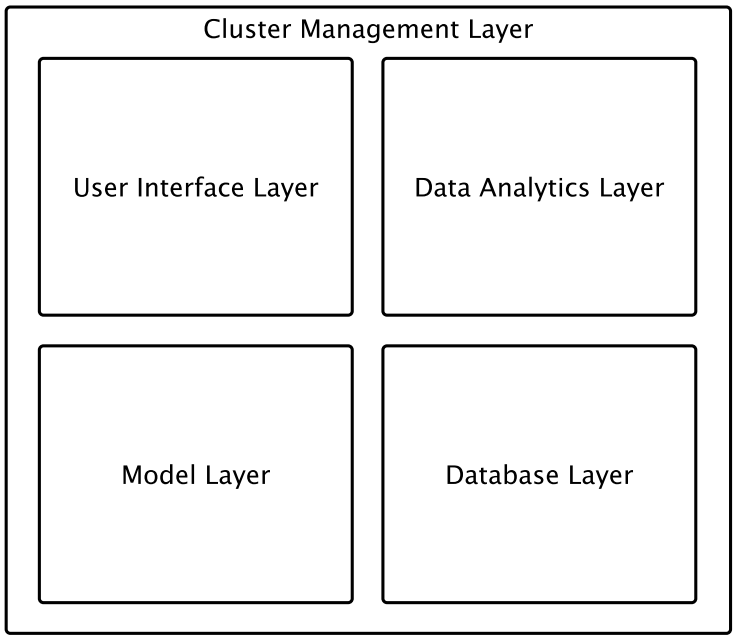


FIGURE 1. Coarse-grain architecture for ScalABM.