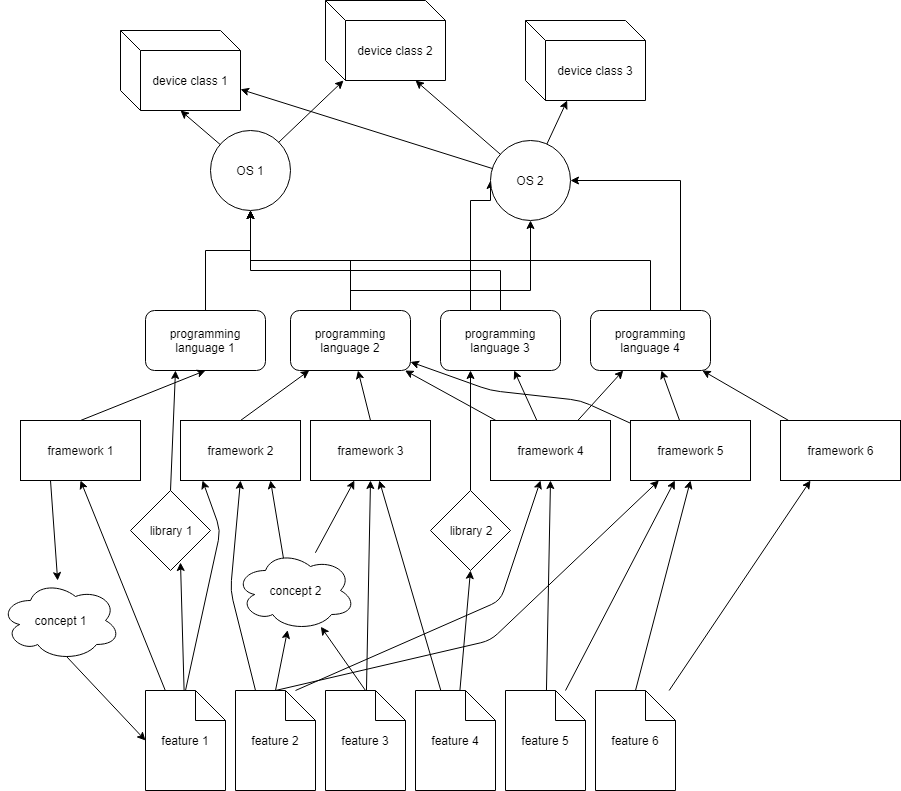
Adaptive game engine

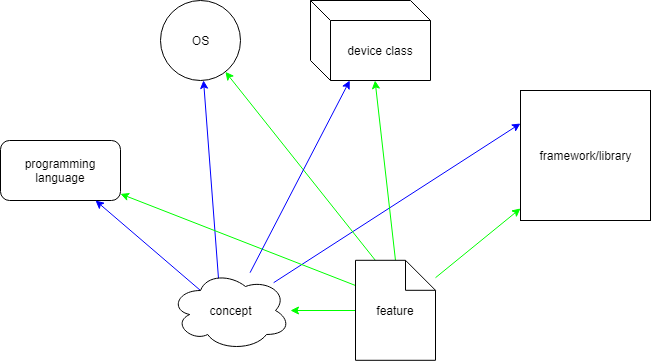
Choosing the means of implementation

By Vokovyy Pavlo

## Basic concepts

Means of implementation consist of different things that provide our project any kind of functionality. It includes, devices (to run the result), operating systems, libraries, frameworks and concepts. The structure is represented by the following diagram:

As you can see, this picture does not show all the dependencies. Some features are defined not by the framework, but by the language itself (like pointers in C++), by the system (like scheduling algorithms and standard library capabilities), or even by the hardware (touchscreens, gyroscopes, etc.). To be graphical, these are all the things that might give us access to a specific feature:



The concept here can be defined as a set of features, bounded by some specification.

# Essential/desired Features

The goal is to find the key components in this scheme that will remain invariant through the first steps of creating the actual application. The most stable of all of the elements shown above are features, so let’s start with them.

Each feature is basically a thing that we must/want be able to do to make the working application. Let’s start from the beginning by searching for needed features in the project description.

Statement 0: We need to create a game with its own generated (and sensitive to any modifications) game world and storyline.

Branch 1: Plot generation (adaptive to user input)

**Choice:** choose the plot generation concept.

Option 1: Get storylines from a database – makes it hard to find a plot resembling all the user’s modifications

* Option 2: Use database information along with plot generating algorithms.

Statement 1: We need to create algorithm to generate plot dynamically from multiple database assets and user input.

Added features: Device with decent computational power. Ability to store data on the device as well as download (large) amount of data from external servers.

**Choice:** choose the plot generation algorithm.

Option 1: Model the game world and run the plot-generating simulation – may use a lot of computational power and might be too complex to create interesting ideas.

Option 2: Combine plot by using the parts of existing plots in the database. – Either requires to have an infinite amount of plots or the parts might not stick to well. Hard to modify as well.

Option 3: Just surf the internet. – Tremendous usage of data and computational power + risk of getting inappropriate info.

* Option 4: Combine all of the above.

Statement 2: We need to create a model of a game-word that corresponds to user input and uses external data to generate complex interactions from specialized database (either local or external) and multiple reasonably-reliable general knowledge resources.

Added features: Painless interaction with different JSON APIs and external databases. Ability to use semantic analysis.

**Choice:** choose the way to simulate the game environment.

Option 1: Use a game engine – Game engines are not designed to build plots and not easily changed.

Option 2: Use a multi-agent framework.

Option 3: Use human resources. – Back to the idea of putting a bunch of test rats inside to make the program work? Magic & Community gathering are not the things developers want to be bothered with before the first release.

* Option 4: Just code it from scratch. – No comments.

Statement 3: We need to simulate the game using concept of multi-agent system to retrieve plot data.

Added concepts: Multi-agent system. (agent, interaction).

**Choice:** choose the storyline-development process.

Option 1: Full rebuild (according to a user input) – “And when I changed the colour of that stone it generated completely different story” ...

Option 2: Receive initial user input-> build the plot-> enable the “manual changes” – “Several days later….”

Option 3: Partial rebuild to fit the changes in user input in the story.

* Option 4: All of the above – No comments.

Statement 4: We need to use/create toolbox to communicate with user during the storyline creation process.

Added features: Ability to efficiently deliver the generated plot to the user in the easy to understand and modify form. Version control.

**Select:** select the basic formats to export the generated storyline.

Option 1: Compatible with popular game engine – is not required. Can be implemented as an additional module later.

* Option 2: Unique format to be used by the current game engine (or converted to any other)

Option 3: Textual (Semantic) easy-to-read format – is not required. Can be implemented as an additional module later.

Option 4: All of the above

Statement 5: We need a mechanism to collect useful data (storyline) from the simulation and export it in a specific format.

Added requirements: Ability to monitor the multi-agent system during the simulation process. Ability to save data in a custom/formalized (JSON, XML) format.

**Select:** select the distribution of computational recourses

Option 1: Single PC – can be slow and the efficient data transfer from the dedicated database might be the issue.

* Option 2: Single PC+SERVER -

Option 3: distributed between multiple computers – unnecessary complicated (for now).

Option 4: All of the above

Statement 5: We might want to extend our multi-agent system on the server to minimize the server-client data transfer.

Added features: (desired)agent communication mechanism through the web. Support by popular server language(s).

# Frameworks/libraries.

What do we know:

1. Framework might save us from reinventing the bicycle.
2. Two or more multi-agent frameworks is messy.
3. No framework and tons of small libraries might be messy to.

There are tons of multi-agent frameworks you can find. But they are not found easily. In my opinion huge role in this phenomenon plays the fact that those frameworks are hardly popularized due to the small multi-agent developer community. That is also the reason they might be not well documented. According to the <http://jasss.soc.surrey.ac.uk/18/1/11.html> the list there was 24 relatively popular multi-agent frameworks.

Throughout the development cycle the requirements to the multi-agent framework will change. Let’s break the choice to 2 stages. To avoid large lists initial filters will be placed to reduce the quantity of frameworks to choose from.

### Stage 1: early development/ test framework (to test new features and shape the idea)

FILTER 1: The framework should be easy to use (it must be easy(quick) to implement different features to test them (before implementing into production version)

FILTER 2: It must feature enough tools to debug and analyse the process.

FILTER 3: It must have documentation (that makes learning curve less horizontal)

### Stage 2: production builds.

FILTER 1: The framework must be free to use (No commercial license permitted)

FILTER 2: It must be light and modular enough (performance is crucial).

FILTER 3: It should have at least some documentation to keep developers sane.

// FILTER 4: It should be available for JS language or at least for any major server-side languages.

FILTER 5: It should be extendible and tweakable – implemented for a popular programming language (not highly specialised like NetLogo or GAMA)

FILTER 6: It should be implemented for JAVA language (easily swappable with another framework in case it won’t meet some of the requirements in the future. 90% of frameworks are implemented for JAVA and that might be the best choice to extend usage sphere on mobile devices).

Early stages Production

1. Agent factory
2. AgentBuilder Commercial license
3. AgentScape
4. Aglobe
5. Anylogic Commercial license
6. Cormas No JAVA
7. Cougar
8. CybelePro Commercial license
9. EMERALD
10. GAMA No JAVA
11. INGENIAS
12. JACK Commercial license
13. JADE Each agent in separate thread
14. JADEX
15. JAMES
16. JAS
17. Jason
18. JIAC
19. MaDKit supports C++
20. MASON
21. NetLogo No JAVA
22. Repast No true parallelism (good?)
23. Sesam Discontinued
24. Swarm Mediocore performance
25. EVE No visualisation and extensive docs WEB-oriented (http/JSON com.)
26. ***Agent Factory*** Framework ([Russell et al. 2011](http://jasss.soc.surrey.ac.uk/18/1/11.html#russell2011)), an open source collection of tools, platforms, and languages that support the development and deployment of multi-agent systems. The framework is broadly split into two parts: support for deploying agents on laptops, desktops, and servers; and support for deploying agents on constrained devices such as mobile phones and sensors. The former support is realized through Agent Factory Standard Edition (AFSE), and the latter support is realized through Agent Factory Micro Edition (AFME), a light weight agent platform that has been designed to work with J2ME-CDLC MIDP2.0. Currently, Agent Factory is used in a number of projects including mobile computing, robotics and other, e.g. the MiRA- Mixed Reality Agents project ([Holz et al. 2011](http://jasss.soc.surrey.ac.uk/18/1/11.html" \l "holz2011)).
27. ***AgentScape*** ([Oey et al. 2010](http://jasss.soc.surrey.ac.uk/18/1/11.html" \l "oey2010)) agent platform, on the other hand, has been designed to support the design and deployment of large-scale, heterogeneous, secure, distributed agent systems. Within AgentScape, agents are active entities that reside within locations, communicate with each other and access services. The AgentScape approach to management is targeted to scalability and autonomicity. The above allows AgentScape to be a promising agent platform for studies including large-scale distribution and heterogeneity.
28. ***AGLOBE*** ([Sislak et al. 2006](http://jasss.soc.surrey.ac.uk/18/1/11.html" \l "sislak2006)) is an agent platform designed for testing experimental scenarios featuring agents' position and communication inaccessibility, but it can be also used without these extended functions. The platform provides functions for the residing agents, such as communication infrastructure, store, directory services, migration function, deploy service, etc. Communication in AGLOBE is very fast and the platform is relatively lightweight. AGLOBE platform is not yet fully compliant with the Foundation for Intelligent Physical Agents' (FIPA) specifications, e.g. it does not support inter-platform communication (communication with other agent platforms is not supported yet). This interoperability is not necessary when developing closed systems, where no communication outside these systems is required (e.g. agent-based simulations). AGLOBE is sponsored by US Air Force and it is suitable for real-world simulations including both static (e.g. towns, ports, etc.) and mobile units (e.g. vehicles). In such case the platform can be started in extended version with Geographical Information System (GIS) services and Environment Simulator (ES) agent.
29. ***Cougaar*** ([Helsinger & Wright 2005](http://jasss.soc.surrey.ac.uk/18/1/11.html" \l "helsinger2005)) follows a Cognitive Agent Architecture and is a DARPA-funded open-source agent platform that offers special support for logistics problems. The platform is not FIPA-compliant. It facilitates the development of agent based applications that are complex, large scale and distributed. Cougaar's cognitive architecture is another promising feature since the majority of the available platforms does not handle with such issues despite the fact that they are important for simulating human thinking and acting.
30. ***EMERALD*** ([Kravari et al. 2010](http://jasss.soc.surrey.ac.uk/18/1/11.html" \l "kravari2010)) is a quite new implementation framework for interoperable reasoning among agents in the Semantic Web, by using third-party trusted reasoning services. The advantage is that every agent can exchange its position justification arguments with any other agent, without the need for all agents to conform to the same kind of rule paradigm or logic. It is built on top of JADE (presented below) and it is fully FIPA-compliant. Moreover, EMERALD was involved in cross-community interoperations such as in Kravari et al. ([2012](http://jasss.soc.surrey.ac.uk/18/1/11.html#kravari2012)). It supports a variety of logics and languages such as Java, JESS, XML, RDF, RuleML and Prolog. Additionally, EMERALD is the only agent platform that supports trust and reputation mechanisms in order to support trustworthiness and efficient decision making in the multi-agent system. It has been used so far in studying how agents act on behalf of their users in cases such as trading.
31. ***INGENIAS Development Kit*** ([Gómez-Sanz & Pavón 2004](http://jasss.soc.surrey.ac.uk/18/1/11.html#gomez-sanz2004)) is a tool for developing Multi-Agent Systems that supports the INGENIAS methodology. INGENIAS promotes a model driven approach based on the use of INGENME (INGENIAS Meta-Editor), a tool for producing self-contained visual editors for languages defined using an XML file. INGENME is used to produce a visual editor for Multi-Agent Systems where the systems' specifications are processed to produce programming code, html documents, or other required products. INGENIAS addresses roundtrip engineering issues as well, by a concrete folder structure and a code-to-specification information migration tool.
32. ***JADE*** ([Bellifemine et al. 2003](http://jasss.soc.surrey.ac.uk/18/1/11.html" \l "bellifemine2003)) is a framework fully implemented in Java. It simplifies the implementation of multi-agent systems through a middle-ware that claims to comply with the FIPA specifications. The agent platform can be distributed across machines (which do not even need to share the same OS) and the configuration can be controlled via a remote GUI. The configuration can be even changed at run-time by moving agents from one machine to another one, as and when required. JADE is completely implemented in the Java language and the minimal system requirement is version 1.2 of JAVA (the run time environment or the JDK), hence it can be adapted to be used on devices with limited resources such as mobile phones. JADE is industry-driven and currently it is the most popular FIPA-compliant agent platform in academic and industrial community. It is a free, open source and stable software distributed by Telecom Italia, the copyright holder. Since May 2003, a JADE Board has been created that supervisions the management of the JADE Project. Currently the JADE Board lists 5members: Telecom Italia, Motorola, Whitestein Technologies AG, Profactor GmbH, and France Telecom R&D.
33. ***Jadex*** BDI Agent System ([Braubach & Pokahr 2013](http://jasss.soc.surrey.ac.uk/18/1/11.html" \l "braubach2013)) follows the Belief Desire Intention (BDI) model and facilitates easy intelligent agent construction with sound software engineering foundations. It allows for programming intelligent software agents in XML and Java. The Jadex research project is conducted by the Distributed Systems and Information Systems Group at the University of Hamburg. The developed software framework is available under GNUs LGPL license, and is continuously evolving. Jadex has been put into practice in the context of several research, teaching, and industrial application scenarios some of which are described in its website. It has been used to build applications in different domains such as simulation, scheduling, and mobile computing. For example, Jadex was used to develop a multi-agent application for negotiation of treatment schedules in hospitals ([Braubach et al. 2014](http://jasss.soc.surrey.ac.uk/18/1/11.html" \l "braubach2014)). In the latest version, the programming model of Jadex is based on the notion of active components that are conceptually based on SCA (service component architecture). This allows for designing an application as hierarchical decomposition of components interacting via services and thus helps making complexity controllable. Active components extend SCA in several directions as it is intended to work in concurrent and dynamic distributed systems.
34. JAMES
35. JAS
36. Jason
37. JIAC
38. MaDKit
39. Repast features C++ fast lib and java simulation toolkit
40. MASON -

JADE – basic solution

MASON || REPAST || MaDkit– simulation

AgentScape