TAMPEREEN TEKNILLINEN YLIOPISTO

SAMI TURUNEN

PRODUCTIZATION OF AN HTML5 AGENT FRAMEWORK

Master of Science Thesis

Examiner: professor Kari Systä
Examiner and topic approved in research council meeting of Computing and Electrical Engineering 5th of May 2014

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
TURUNEN, SAMI: Productization of an HTML5 agent framework
Master of Science Thesis, 58 pages, 0 Appendix pages
March 2015
Major: Software engineering
Examiner: Professor Kari Systä
Keywords: Mobile Agents, productization, installation guide, programming guide

Mobile agents are autonomous software programs, which can move in network and gather information, communicate with each other, and change information. Migration means that mobile agents transfer from one computer system to another. Agents have a state that contains variables and functions which are needed to run agent. State contains also information about the execution state of the agent.
Productization is about making software easy to use trough installation instructions, configuration scripts, and end user documentation. The purpose of this work was to prepare the agent platform for publishing. This thesis presents installation instructions for the agent platform. A configuration script have been made for changing the URLs used by the applications. Programming guide helps novice user to develop mobile agent applications with the agent platform. To make publishing easier, open source licenses, software hosting facilities, and data management systems have been studied.

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
TURUNEN, SAMI: HTML5 agenttisovelluskehyksen tuotteistaminen
Diplomityö, 58 sivua, 0 liitesivua
Maaliskuu 2015
Pääaine: Ohjelmistotuotanto
Tarkastaja: Professori Kari Systä
Avainsanat: Liikkuvat agentit, tuotteistaminen, asennusohje, ohjelmointiohje

Liikkuvat agentit (mobile agents) ovat autonomisia ohjelmia, jotka liikkuvat tietoverkossa ja keräävät tietoa, kommunikoivat toisten agenttien kanssa, ja vaihtavat tietoa. Termi *siirtyminen* (*migration*) tarkoittaa mobiiliagenttien siirtymistä yhdestä tietokonejärjestelmästä toiseen. Agenteilla on *tila*, joka sisältää muuttujat ja funktiot, joita tarvitaan agentin suorittamiseen. Tila sisältää myös tietoa agentin suorituksen tilasta.
Tuotteistaminen tarkoittaa tuotteen tekemistä helppokäyttöiseksi asennusoppaiden, konfigurointiskriptien, ja loppukäyttäjälle suunnatun dokumentaation avulla. Tämän työn tarkoituksena oli valmistella agenttialusta julkaisua varten. Tämä opinnäytetyö esittelee asennusohjeet agenttialustalle. Työssä on kirjoitettu konfigurointiskripti agenttisovellusten käyttämien URLien vaihtamista varten. Ohjelmointiopas auttaa kokematonta käyttäjää kehittämään mobiiliagenttisovelluksia agenttialustan avulla. Julkaisun helpottamiseksi open-source lisensseihin, ohjelmistojen isännöintipalveluihin, ja tiedonhallintajärjestelmiin on tutustuttu.

PREFACE

I started to study the agent platform on spring 2014. I began by studying mobile agents and reading the agent platform code. I wrote installation guide mostly in June. Programming guide, configuration script, and configuration-file modifications were done in autumn.

I want to thank my supervisor Kari Systä for reading and commenting the thesis and giving technical guidance.

In Tampere, on the 5th of February, 2015

Sami Turunen

INDEX

1 INTRODUCTION

The purpose of this work was to study productization and prepare a mobile agent platform for publishing by writing installation and programming guides for the platform. Open Source licenses were studied. Software hosting facilities were searched and data management systems for source code publishing were studied.

The initial version of the agent platform described in this thesis was made by professor Kari Systä at Tampere University of Technology on spring 2012. Version used in this thesis is a second iteration of agent platform that was made by Laura Järvenpää on spring 2013.

Problems for this thesis were: What is productization and especially installation and configuration during productization? How to write a good installation guide: How to make installation of the agent platform easier? How to write a good programming guide: How to make programming agent applications with the agent platform easier? What kind of open-source licenses are there and what are their biggest differences? Where to publish mobile agent platform source code?

Chapter 2 contains general information about mobile agents and examples of mobile agent platforms. Mobility models, communication, and security of mobile agents are briefly described. Chapter 3 gives an overview of mobile agent platform used in this thesis. Chapter 4 contains information about open-source licenses, software hosting facilities, and source code publishing places. Chapter 5 describes terms related to productization and gives an example of software productization process. Chapter 6 contains installation instructions for the agent platform, and a description of an installation- and a configuration-script for the platform. Chapter 7 describes the most important functions and variables of generic and application agent. This chapter also includes a tutorial for creating a simple agent application. Chapter 8 is for evaluation of the installation- and programming guides. Chapter 9 is for conclusions.

2 MOBILE AGENTS

2.1 General information about Mobile Agents

This chapter is mostly based on the book Mobile Agents: Basic Concepts, Mobility Models, and the Tracy Toolkit [1].

2.1.1 Mobile agents

Just like any computer program, a software agent performs tasks given by user in a restricted environment. The software agent can be a program, component, or object (objects are passive while agents are active). Software agents are autonomous, they have own plan that is generated in accordance with the user given task. No user confirmation is needed every step. Software agents communicate with other agents. Agents react on environment changes, identified events shown by sensors. Agents are initiative and plan actively.

Characteristics of mobile software agents are: Agent knows its owner (the user who starts the Agent), his or her preferences and learns by communicating with its owner. Itinerant agents search network efficiently by moving in network to service or information provider. Agents can work asynchronously while the user is offline. Agent reports results of its work to user trough different communication channels. It can suspend execution on arbitrary point and transfer itself to another computer system. This is called migration. At destination computer system agent's execution is resumed. Mobile agents can be considered as additional design paradigm, supplement of traditional techniques.

Mobile agents:

- Are used in wide area and heterogeneous networks where assumptions of reliability of computers or security of network can't be made.
- Migration is initiated by agent or the programmer of agent, not operating system or middleware.
- Migration is done to access resources available only at other servers.
- Can migrate more than once. This is called multi-hop ability.

State means information about the execution state of the agent. The execution state can be information from within the underlying (virtual) machine about call-stack, register values, and instruction pointers. Agent server or agency means execution environment for agents. Agency:

- Provides agents services, communication, and migration to other agencies.
- Controls execution of agents and protects against malicious agents.

A **mobile agent toolkit** [1] is a specific project or product. Examples of mobile agent toolkits are IBM aglets, IKV Grasshopper, and Tracy.

Reasons to use mobile agents are [1]:

- *Task delegation*, which helps to avoid the information overload, because user can focus on other things while agents perform tasks for user.
- *Asynchronous processing*. Mobile agents are not dependent on network connections. Network connection is only needed for the first migration. This ensures that agents can be started from mobile devices that offer only limited bandwidth.
- *Adaptable service interfaces*. Mobile agents offer a chance to design a client-driven interface that is optimized for the client, but also adaptable to different server interfaces.
- *Code-shipping versus data-shipping*. Instead of transferring data to the client (data-shipping), it can be transferred to the location of the data (code-shipping) by means of mobile agents. Only the relevant data is sent back to the client. This reduces network traffic and saves time.

Benefits of mobile agents listed in [5] are:

- Mobile agent can reduce network load by preprocessing sensor data.
- Agents overcome network latency.
- Agents encapsulate protocols; agents are good tools for introducing new protocols or data formats.
- Agents execute asynchronously and autonomously. There is no need to generate network traffic for every execution.

Domains where agents are noticed useful are electronic commerce and information retrieval. One advance of agents is that they can filter sensor data. [1]

2.1.2 Migration

Migration means transferring mobile agent from one computer system to another. As compared to the client-server applications migration saves network bandwidth by moving the code close to the data. Mobile agent is executed as part of the mobile agent server software. It controls execution of agents and provides functionality for agent communication, agent control, security, and migration. In Mobile Agents-book [1] mobile agent server is called agency. Agency that holds the agent's code is code server, which usually is the home agency. Mobile agents consist of code, data(variables), and execution state. Mobile Agents-book names current agency as sender agency, S and other agency as receiver agency, R. The migration protocol defined in the book has six steps (see the figure 2.1):

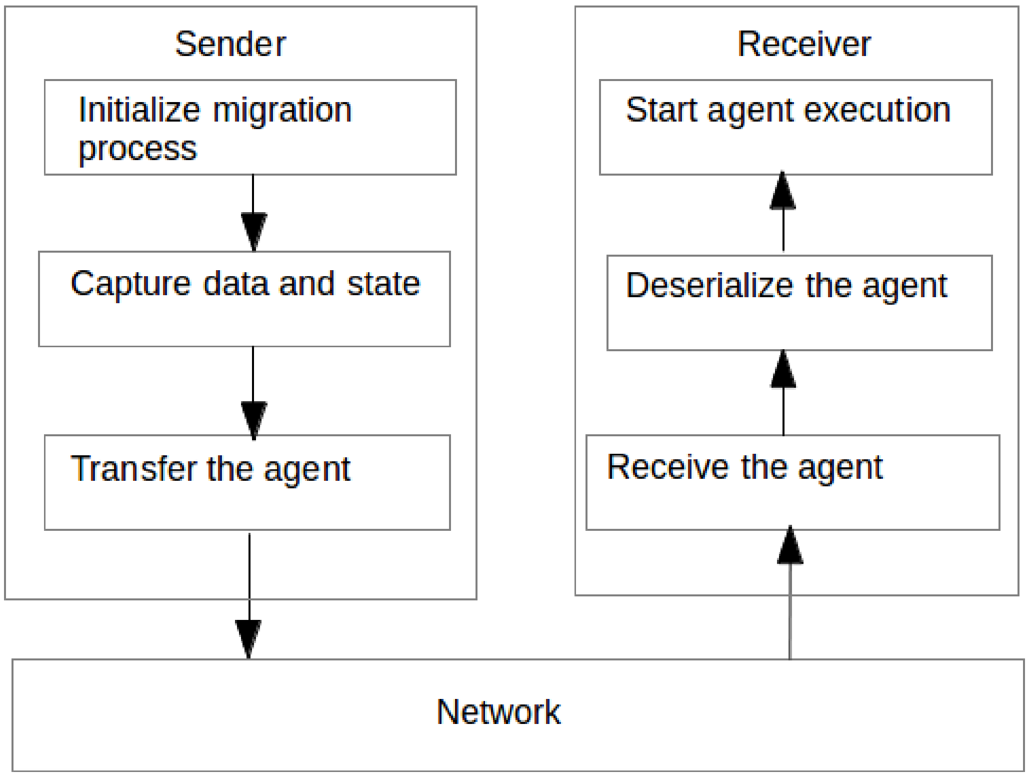


***Figure 2.1**: Migration process [1]*

- S1: Suspend the execution thread, initialize the migration process.
- S2: Capture agent's data and execution state. Serialization of the current state of all variables. Current values are written to an external persistent representation, which can be a memory block or a file. Agent's state is stored in same place. Outcome is a serialized agent, a flat byte stream.
- S3: Transfer the agent: Serialized agent is transferred to the receiver agency using a migration protocol.

- R1: Receive the agent: Serialized agent is received using a migration protocol. Receiver agency checks whether the agent can be accepted based on the information about the owner and the sender agency.
- R2: Deserialize the agent: Variables and the execution state are restored from the serialized agent.
- R3: Start agent execution in new thread. Receiver agency starts a new thread of control. The agent's code is needed to do this. Also the code has to be transferred to the receiver agency. It can be loaded from the agent's home agency or its code server. [1]

2.1.3 Mobility models of Aglets and Grasshopper

Mobility model [1] defines three views on migration issues:

- 1. User's view: How migration is initiated.
- 2. Agent's view: How data and code are relocated in the network.
- 3. Network's view: How code and data are transferred over the network.

User's view focuses on phases S1 and R3 of agent migration. Agent can be immediately started at a remote agency and not at home agency, if other agencies support this by providing a communication interface for this purpose. Place of code source (place from which code can be loaded) is in Java CLASSPATH -variable or files are in the filesystem and parameter is given during creation. Each class file should be stored separately.

Useful data types for mobile agents are:

- 1. *Proxy*: Mobile Agents-book[1] names this data type proxy, because there is a proxy object on each agency that is part of the serialized agent and that is responsible for transparently forwarding modifications to the home agency.
- 2. *Static*: Static data is nonmobile and remote access is not possible. This data type is common for files or graphical user interfaces, whenever they are not of type proxy.
- 3. *Moving*: Moving data is mobile, and the source has been removed. Data items of this type no longer exist at sender agency after migration.
- 4. *Copying*: Copying is mobile, and the source is not removed. This data type is used for all variables for which the agent has a reference and that are shared with other agents or the agency itself. Modifications are not visible at the original data item at the last agency.

Migration strategies are:

- Push-all-to-next: The code of the agent and the serialized agent are transmitted simultaneously. Connection to the home agency is no longer needed after this one transmission.
- Pull strategy: No code is transmitted with the data transmission. There must be open network connection or a fast way to reconnect to the home agency or the last agency the agent came from.
- Push-all-to-all: The complete code of an agent is sent to all destination platforms agent is going to visit.

Network's view means all aspects related to data transmission. Agent toolkits are using different transmission strategies; for example migration protocol based on RMI or TCP/IP.

An Aglet [8,9] is a Java agent that can move from host to another and communicate with other aglets. In Aglets [1] method *run* starts the agent. *Dispatch* initiates migration. Migration strategy is combination of push and pull strategies.

In Grasshopper [1] method *live* starts the agent. *Move* initiates migration. When error occurs, an exception is thrown. Grasshopper uses class loader for pulling, and it has no push strategy. Agent has a veto right, it has a right to vote against migration.

Transmission aspects: Downloading classes from a nearby server is important, because shorter distance may improve the time needed for downloading.

Migration strategies: Decision of migration strategy depends on network model (structure). In homogeneous network pull-per-unit is a good strategy. In heterogeneous network it is useful to push code in most cases. [1]

2.1.4 Kalong mobility model

Kalong [1] allows agent or it's programmer to define new migration strategies. How Kalong differs from current mobility models:

- 1. Agents have an external state and code is transmitted in code units
- 2. It has code server agency and mirror agency
- 3. It has new class cache mechanism, it can prevent class downloading or code transmission

Kalong's functions to get and set data of external state are *getData* and *setData.* If sender agency is remote agency, all data of external state is transferred with agent's state. *loadData* and *uploadData* are functions for loading and uploading data item of an external state from/to home agency.

Kalong model makes possible less transmission of data items and code. It also increases security; agent can leave important data items to the home agency until they are needed. A code unit consist of at least one Java class (a JAR-file). Code base is the place, where unit can be loaded. Two agents can have different code units even if they belong to the same type. Things that must be known to be able to download code: 1. To which code unit does the class belong 2. From which code base the code unit should be loaded.

Migration process: The parts in which the agent is transmitted are: 1. State, 2. Code units(Java class files).

Types of agencies:

- *Code server agency*: Some code units will be stored at the current agency.
- *Mirror agency*: Keeps information about code and data, can take the role of the home agency.

Code cache: Basis of classes technique digests or hash values to check whether two classes are equal. Prevents transmission of identical code units.

When the agent terminates it has to release all code servers to free resources.

2.2 Examples of Mobile Agent platforms

2.2.1 Radigost

Radigost [2] is a multi-agent platform executed inside of the client's web browser. Radigost is mostly developed in JavaScript and doesn't require third party plug-ins like Java or Adobe Flash. It requires no additional software packages and can be started out-of-the-box. Radigost relies on web workers and web sockets. Web workers enable multi-threading and asynchronous messaging, which means that no timers or scheduling are needed. Radigost agents interact with host environment trough message exchange or included listener object. Radigost supports reactive architecture: agent behavior is based on messages received from the environment or other agents. Each agent has a globally unique identifier, AID, which is a string value. Agents have a state that can be serialized into JSON string. Radigost is interoperable: Radigost agent can interact with other agents in third-party multi-agent platforms. Radigost's runtime performance has been evaluated with several case studies.

2.2.2 Telescript

Telescript programming language [10] is designed for development of mobile agent programs. In Telescript *places* are processes that are inhabited by agents. Telescript agent needs a *ticket* containing address of the place to move from place to another. Ticket also contains definition of how the agent will travel, like the communication protocol. Agents can communicate with each other in *meetings*. To meet another agent, a valid petition has to be presented by the agent initiating the meeting. Agents can be created and accessed by submitting special HTML forms or following hyperlinks. Migration is initiated with *go*-command.

2.2.3 Grasshopper

Grasshopper [28] is a mobile agent development and runtime platform. In Grasshopper two types of agents are distinguished, mobile agents and stationary agents. The Grasshopper agency consists of the core agency and one or more places. Core agency provides a communication, registration, management, security, and persistence services. A place provides a logical grouping of functionality inside of an agency.

2.2.4 Tracy toolkit

Tracy [1] is a mobile agent toolkit made in Java programming language. Advantages of Java mentioned in the Mobile Agents-book [1] are that Java is portable, it has security manager, and it's pointer model doesn't support illegal type casting. Network programming is supported with RMI, remote method invocation, which helps implementing simple mobile agent toolkits. Drawbacks of Java are that it is impossible to obtain execution state of a thread, which means that Java based agents can form only weak form of mobility. Other drawbacks are the lack of resource control and means to avoid denial of service attacks.

Tracy has an abstract class Agent and abstract method *startAgent*. It has a migration protocol SATP based on TCP/IP. Command *go* initiates migration. Agent execution is resumed in receiver agency using Java reflection technique that determines information about classes, their variables and methods during runtime. All classes whose objects must be serialized must implement interface java.io.Serializable. Class variables are not a part of the serialized object. Method name to invoke at destination and classes that agent might ever use are transferred to the receiver agency. Redundant classes are deleted from the code closure before the byte code is collected. Code closure consists of the agent's main class and all the classes that are used for variables, method parameters, method return values, and all local variables of any class of the code closure. Class is searched in the agent's code base, which is defined when launching an agent. All class files are already at the destination agency, so class loader has to look for the byte code only in a local repository, where the incoming classes are stored.

2.3 Communication

Example case where agent communication [1] is needed, is an application in which agent arranges a business trip for a human user. Slave agents are specialized agents to find information about flights, hotels, etc. These specialized agents are instructed by master agent that communicates to the user and monitors the entire process. All these agents exchange information about their tasks and intermediate results. The main problem in getting mobile agents to communicate with each other is how to locate agents that can move autonomously in network. Locating is important so that agent's task can be modified and send a termination signal (when its has become obsolete). It must be possible to detect agents that are orphans to avoid unnecessarily consuming of resources. Orphan agent's owners are no longer interested of it's result or are not available because of a host error. Host error can occur when a master agent's agency crashes.

An energy counter is one solution to avoid unnecessary consuming of resources. Every agent has certain amount of energy which it consumes. One possibility is that the energy counter can be decreased after every migration. Counter could also be decreased for every service access or each message sent. Agent must periodically contact to the master agent or home agency for a new energy. Otherwise the agent runs out of energy and must terminate.

Message passing is a type of communication model that allows agents to send messages to each other. The sender of the message must know the receiver by name and its current location. Point-to-point is a form of message passing in which a single agent sends messages to exactly one receiver agent. In multi-point message passing form group of agents must communicate. It usually uses point-to-multi-point communication technique, in which sender agent wants to send messages to many or all agents of the group.

Communication can be either synchronous or asynchronous. In synchronous communication addresser(sender) blocks it own execution until addressee has answered with a reply. In asynchronous communication addresser sends a message to the addressee and continues its own execution.

Second communication model is information space: Agent writes data to information space and other agents can read it. The most important difference to message passing is that an agent doesn't have to decide which piece of information must be sent to whom.

Full information approach assumes that every agency knows the current location of all agents in the system. This approach is good for delivering messages.

In no information approach no agency has a direct knowledge of the current location of agents. Locating agent for message delivery is difficult. Whether to favor full information approach or no information approach depends of agent: Is the agent mobile and does the agent receive a lot of messages. [1]

2.4 Security

Agent can consume resources of the hosting environment. This can lead to situation where agency can no longer provide it's usual service to the other agents. Agent might try to gain unauthorized access to the agency. This can be solved using Java sandbox technique. Agent can attack other agents. In Java this might be avoided by defining separate class loader to agent.

Malicious agency is an agency that tries to attack mobile agents. Mobile agent must authenticate on every agency and agency must authenticate to the agent. Agency can attack to other agencies communication link. It can tamper or delete data of other agencies. Agency can attack to agent to cause another agency to malfunction. In case of black box attack agent is executed several times changing input parameters and calls to services. One solution to avoid attacks is letting agent migrate only to trusted agencies. Only agents can complain about agencies; agencies have no means to decline a bad reputation. [1]

Example of malicious host is the host for a shopping agent to find the best airfare for a flight with a particular route. To make it's own offer to look the best host could erase information collected by the agent, change agent's route, or terminate the agent. To solve problems with malicious hosts, contracts between the operators of agent platforms could be made or trusted third party could supply trusted hardware.

Agents should be prevented from launching denial-of-service attack. Agent should act only based on information from trusted sources. Agents could be held responsible for their actions and agent behavior could be monitored and logged. [7]

One way to protect mobile agents is that almost all data stored within agent's state or data package must have an expiration date, which agencies can verify, and reject the agent and the agent's data item if necessary. Other way is that when agency has received agent, it verifies the state appraisal functions. These functions verify specific conditions or invariants in the agents state. Agent will have a set of privileges for execution in the agency, or this can result as rejecting the agent. Privileges can be decided based on the agent behavior at runtime (history based decision). Firewall is needed to examine agent before it leaves agency, because it could carry a piece of code with it. Only agent replication can detect an attack and recover the error. [1]

3 HTML5 MOBILE AGENTS

Research papers [5,6,56] and thesis work [3] as main references.

3.1 Introduction

The goal of HTML5 as mentioned in [6], is to “allow the development of complete client-side applications.” HTML5 allows more applications to be run in browser. Browser is increasingly acting as an application platform and applications can store their internal state in the server for future use. In HTML5 agent framework, the executable code can be moved with the internal state of the application. Agents can continue their execution while being stored in the server, and the running applications can be later retrieved back to the browser. The agents are implemented as HTML5 applications, and are therefore called as HTML5 agents. [6]

Mobile agent framework used in this thesis consists of HTML agent base class and agent server. Agent base class works with user interface inside browser or in headless mode in an application server. [3]

Code-on-demand paradigm [33] is used for getting the static files of the application when agent travels in network. Agent server represents mobile agent environment in server side and browser represents mobile agent environment in client side. Implementation of server side execution environment is done with node.js [29].

Like most HTML5 applications, HTML5 agents are composed of two parts:

1. Description of the user interface in HTML, CSS, and image files.
2. JavaScript files describing the executable content and dynamic aspects of user interface.

Generic parts of the agent framework are included into one JavaScript class Agent, and concrete agents are implemented by inhering from this class. Agent can move between different devices, and it is possible to clone agents to create more instances. Agents have variable list that contains the state that has been saved and transferred. The state of the agent is saved during the migration between server and browser. Serialization is done by constructing a JSON-string that contains the agent description. Transferring of agent is done by HTTP. [3,5,6]

More detailed information about classes, functions and variables can be found in chapter 7 Programming guide.

3.1.1 Reference models for mobile agents

According to [56], a mobile agent must contain *an agent model*, *a life cycle model*, *a computational model*, *a security model*, *a communication model*, and *a navigation model*.

Agent model [56] defines the internal structure of the intelligent agent part of a mobile agent. This model also defines the autonomy, learning, and co-operative characteristics of the agent.

Life cycle model [56] defines the different execution states of a mobile agent and the events that cause the movement from one state to another. General structure for *persistent process based life cycle* is described in figure 3.1.

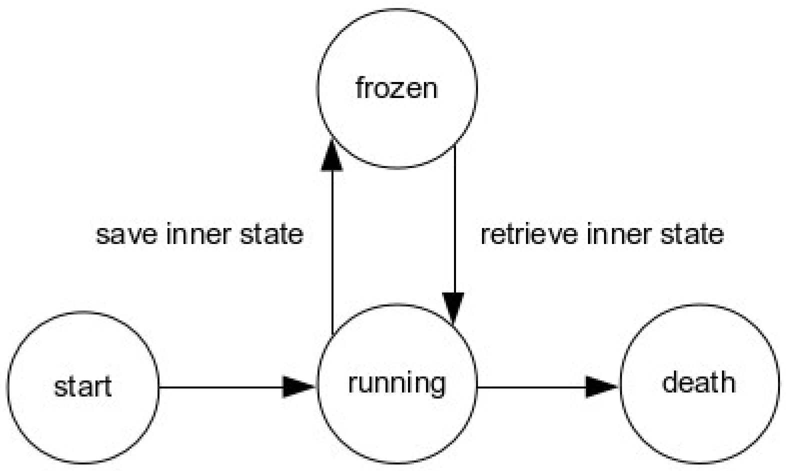


Figure 3.1: *Persistent process based life cycle [56,3].*

Life cycle [56,3] starts with a *start*-state, where the agent is started for the first time. Then the agent is moved to a *running*-state, where a persistent process is executed. Before the migration from one host to another, the inner state of the agent is saved and the agent enters the *frozen*-state. After the migration the inner state of the agent is retrieved and the agent continues execution in the running-state at the point where it left off. After the execution is finished the agent enters a *death*-state, where the process is terminated.

Computational model [56] describes how the execution of a mobile agent occurs when the mobile agent is in *running*-state.

Security model can be split into two areas: Protection of hosts from destructive mobile agents and protection of mobile agents from destructive hosts.

Communication model: According to [6], "Communication model defines means to communicate with the environment and other agents."

Navigation model [56] discovers all aspects of agent mobility from discovery and resolution of destination hosts to the manner in which the mobile agent is transported.

3.1.2 HTML5 agents and mobile agent models

HTML5 agent [3] needs the models for the mobile agent. Agent-, life cycle-, and computational models have complete implementations. Navigation-, communication-, and security models are on the initial stage.

Agent model [3] includes the management of the inner state of the agent.

Life cycle model [3] (main life cycle of the agent) follows the persistent process life cycle described in section 3.1.1. During its life cycle, agent may visit several browsers and agent servers [5]. An example life cycle is described in figure 3.2.

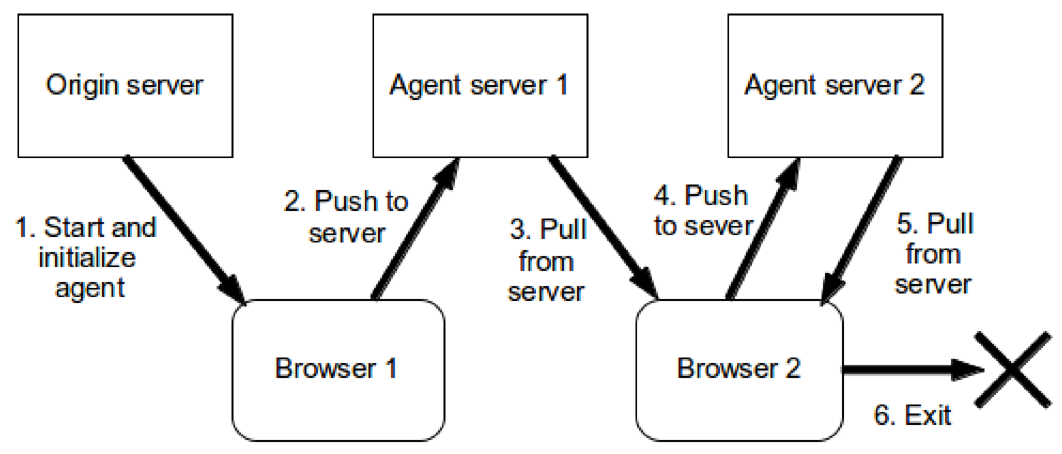


Figure 3.2: *HTML5 agent life cycle [5].*

In step 1 (figure 3.2), agent is started in Browser 1 when it is downloaded from its origin server. Agent is initialized and the execution begins. In step 2 agent is pushed to agent server, where it can continue its execution. The agent server gets the internal execution state of the agent and URL to the application code. In steps 3-5 agent moves from one environment to another preserving its internal state and continuing execution. In step 6, the execution is terminated. [6]

The origin server and agent server are HTTP-servers that can be accessed with HTTP-requests. Agent is fetched for execution with GET and pushed to server with POST. Because agent can run with or without user interface, the execution part of the HTML5 agent needs to be separate from the user interface. [5]

Computational model [3] of application needs to be run in both browser and server. All computation is implemented in JavaScript and the framework is based on event handlers [6].

Navigation model [3] of the HTML5 agent consists of the configuration-file that is downloaded with the agent. In the current implementation, navigation model includes only connections to one agent server and the origin server of the agent. Navigation model also includes the serialization and transfer management of the agent.

Communication model [3] includes communication with the user, third party services, agent server, and another agent application. For agent-to-agent communication, there are three cases [3]:

1. Both agent applications are in separate servers
2. Both agent applications are in agent server
3. One agent is in browser and another is in server

Security model [3] relies in standard security mechanisms of HTML5 applications in browser.

3.1.3 Architecture

Figure 3.3 describes the basic architecture of an HTML5 agent framework and its relationship to application specific implementation. Agent part of the architecture consists of a generic agent and an application agent.

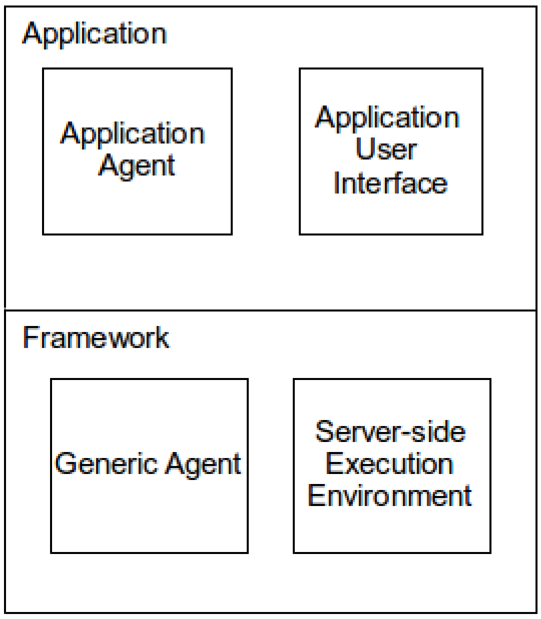


***Figure 3.3**: Basic architecture of framework [3].*

In server, the agent can be accessed trough generic interface provided by generic agent. In browser, the agent can be accessed trough application specific user interface [3].

3.2 Agent server

Core components of the agent server are a HTTP server and a virtual machine executing JavaScript [5]. Agent server consist of server, router and handlers. In agent server dependency injection design pattern [31, 3] is used. The agent server receives the description of the agent in an HTTP POST request [6]. Then the following steps [6] are executed:

1. The executable JavaScript-file is downloaded from the origin server of the agent.
2. The required run-time structures are created.
3. The function *continueWork()* of the downloaded agent is called.
4. A timer to periodically fire *work()*-function is initialized.

When the agent is running in server, it runs in headless mode and the HTML and CSS files are not needed. In headless mode, the central components are the JavaScript files. [5]

3.3 Agent-to-agent communication

Real-time agent-to-agent communication [3] is made with node.js [29] module socket.io [30]. In browser-to-browser messaging, server is always needed. Agents must use communication component to get connection to the agent server. Agent sends information in JSON form to agent server, and agent server passes the information to other agent. Agent-to-agent communication using agent server is described in figure 3.4.

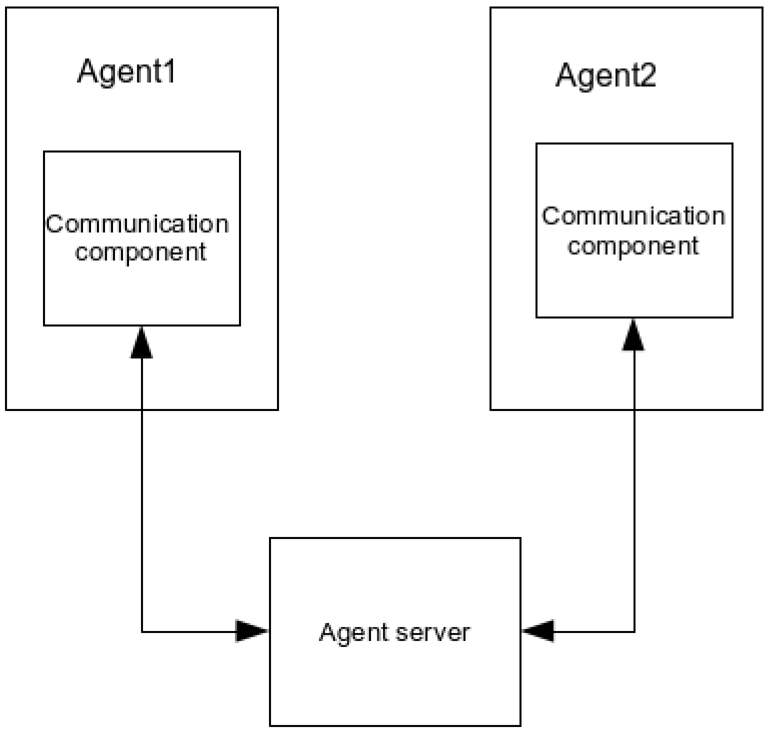


Figure 3.4*: Agent-to-agent communication [3]*

Sender agent and receiver agent must be joined to the same communication namespace to be able to communicate. [3] Agent-to-agent communication was demonstrated with an example (see section 7.5) where agent sends counting results data to another agent.

3.4 Configuring HTML5 agents

Configuration of all HTML5 agents is done by modifying *configuration.js*-file, which includes URLs to:

- The origin server of the agent,

- list view of executing agents in server,
- agent server, where agent is uploaded,
- agent server, that agent uses to communicate [3].

The origin server of the agent maintains and serves all the files. All agent applications have the same origin server. While the agent moves, URL to a resource in the origin server is usually delivered instead of the actual content. [3,5]

In the list view, a list of active agents is shown. List shows the id, URL, and running status of agents.

http://localhost:8891/upload sends URLs to agent code and user interface together with serialized state [5].

In the current implementation, agent server for communication is the same as the server where agent is uploaded. [3]

User of the HTML5 agent framework can change the configuration URLs with the help of a configuration script written for this thesis (see section 6.9). There were hard-coded port number-values in the *fileserver.js*-, *agentserver.js*-, and *loadserver.js*-files. Modifications to the configuration-file have been done to get the port numbers directly from configuration URLs (see section 6.10).

ConfigurationClass() -function should be used when referring to the URLs. A Configuration object can be created by calling constructor *configurationClass* (which will create object that includes specified URLs.) [3]

3.5 Gmonitor

Gmonitor.js is an example agent application that monitors load average of a host and tracks the CPU load of a server machine. In browser, gmonitor shows current load level, min and max values, and the latest load level history graphically. In server history information is still collected. Count of taken samples is shown as running status on the agent server list view. The specialized parts of gmonitor are:

1. Constructor
2. Function *continueWork()*
3. Function *preupload()*
4. Function *work()*.

Draw()-function draws the user interface. *CreateAgentObject()*-function creates agent. *InitExecution()*-function creates an agent object and starts the agent execution in the browser. [3]

In this thesis, Gmonitor was used to test the agent framework installation (see section 6.7).

4 OPEN SOURCE

4.1 Open Source Licenses

Open source licenses [26] ensure that software developers can control the terms under which others can reuse the software they contribute to open source projects. All open source licenses share the principle that anyone should be able to use, copy, and distribute source code and the executable software compiled from it.

Bilen [63] divides software developing companies in technology industry to two groups. The first takes the advantage of open source licenses like Apache [55] and BSD [64] which allow them to use these open source components to develop their own proprietary software. The second group of companies, which stick to GPL [65] and GPL-like licenses, focus on pure open source software development and utilize the full power of outside-in thinking via communities and networks such as universities or other companies.

Apache License [17,26], version 2.0 grants a copyright and a patent license from contributor to users. Redistribution is allowed under following conditions:

1. Recipients of the work must be given a copy of the license
2. Modified files include notices about modification
3. All copyright, patent, trademark, and attribution notices must be retained from the licensed work, excluding notices that don't belong to any part of derivative work
4. If the work includes “notice” text file, derivative work must include this file

Creating and selling customized versions of Apache is allowed. License doesn't give permission to use trademarks. No warranty is given and contributors are not liable for the damages arising as a result of using the work.

The MIT License [17,26] gives no restrictions for dealing with the software. No warranty is given and authors or copyright holders are not liable. MIT license is GPL compatible.

GNU General Public License (GPL) [17,18,26,65], version 2.0 allows to distribute copies of free software, receive source code, change software, or use pieces of software in new programs. When distributing the copies, all recipients must be given all the rights that the distributor has. Recipients must receive or have possibility to get the source code and they must see the license terms. No warranty is given. Copying and

distributing is allowed, provided that copies are published with copyright notice and disclaimer of warranty. Modifying copy of program is allowed under certain conditions:
a) Modified files must have notices that files have been changed and date when changes have been made.
b) Work that contains any part of the program must be licensed under the terms of this license.
c) Interactive program should print or display an announcement including copyright notice, and a notice that there is no warranty and that users may redistribute the program under the conditions, and tell users how they can view this license.

Granting a sublicense to modify and distribute software to third parties not included in the license is forbidden. GPL has a Copyleft restriction, which means that it doesn't allow users to build proprietary software on top of open source foundations. GPL does the most to ensure that software stays open, while other licenses are more restrictive.

LGPL [26] is a lesser restrictive version of the GPL for use with the code libraries that are used in precompiled binary form and do not require access to source code.

BSD licenses [26,64] are very permissive licenses. Unlike the GPL, BSD licenses allow proprietary products to be built and sold from the BSD code without requiring that the source code be given away. BSD licenses are generally GPL compatible.

The Artistic License (Perl) [26] has several differences (compared) to the Apache license. The Artistic license allows changes to the source code if:

- User makes these changes in public domain
- Or user uses only modified code within a company or organization.
- Or user renames modified code so that it doesn't conflict with the standard version.

The Artistic License is GPL compatible.

In **Mozilla Public License (MPL)** [26] modifications must be made freely available to the developer community. This resembles to the GPL. Larger pieces of software are not required to make source code available. Developers can improve the open source code base without giving away their own property. This differs from the GPL. MPL is not GPL compatible on its own, but there is an effort to relicense all Mozilla code under a triple license with the GPL and LGPL.

Dual licensing [26] means that sellers offer option of choosing either a GPL option or more conventional commercial license.

4.2 Choosing a license

In [53], choosing a license is decided based on whether the user wants a copyleft license or non-copyleft license. Recommended copyleft licenses are the GPL-3.0 [54] or the AGPL-3.0 [41]. If granting patent rights from contributors to users is important, recommended non-copyleft license is Apache-2.0 [55]. If not, MIT is very permissive and simple as well as BSD. Apache, MIT, and BSD-licenses are proprietary-compatible licenses, they allow the covered code to be used in proprietary programs. All these licenses are widely used and well recognized. People don't have to read the legalese in order to use the code, because they have already done that. It is easier for people to entry for this kind of project. Licenses are high quality and revisions of previous versions of themselves. [53]

According to Rosen [57] for commercial companies the discussion of which open source license to use often centers one or both of following issues:

1. "How can we make money from distributing this software under an open source license? In essence, can our license help us sell free software?"
2. "How can we prevent others from making money unfairly from our open source software? This is the so-called free- rider issue, where licenses reap all the benefits of others' work with no return obligations."

4.2.1 Making money from Open Source

Customers are often willing to pay for brand-name software [57], particularly if it comes with support and other benefits. Names and reputations of the licensors are excluded from several open source licenses. Licensors can protect their names and reputations for personal profit. Krishnamurthy [66] introduces three ways on how distributors can make money:

1. They provide the product on CD rather than as an online download. Most people don't want to download the product from a web site. There is money to be made selling the product in CD form.
2. There is money to be made in services such as support for installation and answering technical questions. The creators of the software can be considered the best suited to provide support.
3. Distributors can act as application service providers and help their clients get the latest version of the product as well as updates on the product.

Licensor of open source software is always free to license his or her software under other terms and conditions. Licensee may contact the licensor to determine if the software is also available under a different license. [57]

4.2.2 The free rider problem

All licensees are free to copy and create derivative works without payment of royalties to the licensor. If it is important to discourage free riders who create and distribute derivative works, then a reciprocal (for example GPL) license is often more effective than an academic (for example BSD) license. “At least with reciprocal licenses, everyone is a free-rider of everyone else's distributed derivative works, because that software is licensed under the same license.” To be able to avoid free riders, adopting one of the non open source licenses should be considered, or try instead to make money with a proprietary software distribution model. [57] According to Maxwell [62], increasing the number of potential contributors help to avoid the free rider problem, because “The larger the group of potential contributors, the better the odds that someone individuals' personal cost/benefit calculations will lead them to participate and thus contribute.”

4.2.3 Motivations for sharing Open Source software

Maxwell [62] discusses about reasons why people develop and share open source software. One reason is altruism, the desire to be helpful to the open source community. Some people rely less on financial rewards than on the positive feelings they get from helping others. Programmers get the feeling of excitement and accomplishment from writing a program or solving a difficult problem. A key reason why programmers are attending to working with shared software is that they are attempting to solve a technical problem of their own. “If an individual shares code that provides benefits to others, that individual may benefit, both now and in the future, as others share code that they have written.” Another reason why individuals participate to open source development is increased reputation among their peers. This can lead to promotion or give possibilities to getting a better job offers based on the enhanced reputation. One reason to participate in open source development is to improve one's programming skills.

4.2.4 Out-Licensing

Licensors decide what license to use for their open source software. If possible, an existing template license should be used. Typical considerations when out-licensing software are [57]:

- The key licensing factor is whether to use a reciprocal (GPL, MPL, CPL [60], OSL [59]) or an academic (BSD, MIT, Apache, AFL [58]) license. “As a licensor, do you want to be able to benefit from improvements made by others? Do you want derivative works created by your licensees to be distributed under the same license so that you can incorporate their improvements into your own software?”
- If reciprocal license is selected, consider the scope of reciprocity obligation. Licenses like GPL contain vague provisions about derivative and collective works. Licenses like the MPL have a more narrow definition of derivative

works, requiring only files that are changed to be distributed under the MPL. The CPL or OSL leave the term derivative works to be defined by the courts under copyright law.

- Scope of any patent licenses that will be granted should be considered. Licenses with explicit patent grants are the Mozilla, CPL, and OSL/AFL licenses. For these licenses you should decide whether you wish to allow your patents to be used for creating derivative works. (These licenses have subtly different patent grants.)
- Are you prepared to grant a warranty of provenance (OSL/AFL licenses, and similar “representations” in the MPL and CPL licenses), or do you prefer to disclaim all warranties? (In some countries you may have to accept warranties regardless of what your license says.)
- Do you want a defense against patent infringement lawsuits and should the defensive strategy terminate only patent licenses (Mozilla and CPL licenses) or both copyright and patent licenses (OSL/AFL licenses) for patent infringement claims? “Is it sufficient to mandate an express condition that the software cannot be distributed if there is a patent infringement claim against the software (i.e., the GPL license)?”
- “Do you want your license to be interpreted under copyright law only (i.e., the GPL) or under both copyright and contract law (i.e., almost all other open source licenses)?”

Rosen [57] advices consulting one's attorney before actually crafting or selecting a license.

4.3 Things to consider when applying an open source license

Hecker [61] addresses things that company has to do to implement an open source strategy:

- If your open source product shares a common source code base with others of your products that will remain proprietary, make sure that open-source development can proceed without complicating your other internal development efforts. (This may require licensing considerations and appropriate modularization.)
- If your product includes technology licensed from the third parties, you must treat third-party code specially to create a releasable product. Ways are removing code entirely, seeking permission to include third-party code, or replacing it with

open source code providing equivalent or similar functionality. The presence of third-party technology can also affect on selecting an open source license.

- To ensure that the source code is ready for public distribution, remove or revise inappropriate language and comments intended for internal viewing only. Bugs should be searched from the source code and fixed before releasing the source code. Woods & Guliani [26] address that code should be organized in a way that invites understanding, and that reveals at least some organization. Code should be modularized and grouping of the modules together should have considered. A naming convention should have been rigorously adhered to.

If the code files include code that is already licensed under certain license, like *XMLHttpRequest.js* that is licensed under MIT-license, it is important to check that those licenses are compatible with the chosen license.

4.4 Open Source Software Hosting Facilities

SourceForge [15] is a website for finding, creating and publishing Open Source software for free. It offers custom project page and download status information. SourceForge platform is Allura [52], which is released under the Apache license.

Launchpad [16,21] uses Bazaar version control system [51] for code hosting. Launchpad includes bug tracking, code review, and translation tools. Launchpad source code is released under AGPL-3.0 [41]. Launchpad offers users possibility to publish project files for download. At first, the release must be recorded in Lauchpad.

In **GitHub** [19,20] public repositories are used to share Open Source software. GitHub offers an option to include a software license when creating a new repository. GitHub repositories can be accessed from Git [48] and Subversion [50] clients.

Google Code [22] offers project hosting with Git [48], Mercurial [49], and Subversion [50]. It offers two gigabytes of storage space. Code browsing and code review tools, issue tracker and project wiki are also included.

BitBucket [23] offers free source code hosting for Git [48] and Mercurial [49]. It offers free, unlimited private repositories for up to five developers. Features include pull requests and code reviews.

Codehaus [24] is environment for building open source projects. Codehaus accepts only projects that have business-friendly licenses. Permitted and prohibited licenses are listed in licenses page [67]. Projects hosted on the site are selected by reviewers, existing Codehaus committers.

Gitorius [47] provides free code hosting for open-source projects that use Git [48]. Gitorius is free software licensed under GNU Affero General Public License version 3 [41].

4.5 Data management systems

This section introduces publishing systems for open source software. These systems are presented here as an option for project code hosting places presented in section 4.4.

DSpace [37] is an open source repository application for academic, non-profit, and commercial organizations building open digital repositories. It is free, easy to install "out-of-the-box" and customizable to fit the needs of the organization. It preserves easy and open access to all kinds of digital content including text, images, and datasets. DSpace's code is currently licensed under the BSD license. DSpace version 4.2 can be downloaded from the files area in SourceForge [38] or via GitHub [39]. DSpace documentation is available at Duraspace [40].

CKAN [34] is a data management system that makes data accessible by providing tools to streamline publishing, sharing, finding and using data. It is aimed at data publishers wanting to make their data open and available. CKAN is licensed under the terms of the Affero GNU GPL v.3.0 [41]. CKAN offers several ways for adding and editing data: Directly via the web interface, using CKAN's JSON API [35], or via custom spreadsheet importers. Searching includes keyword search, faceting by tags, and browsing between related datasets. CKAN provides metadata (title, description, license) for each dataset. It offers geospatial features covering data preview, search, and discovery. For communicating and collaborating on data, CKAN offers comment and discussion features, RSS/Atom feeds for changes and revisions, and possibility to follow these changes. CKAN offers data storage and a resource page for uploading data resources.

GO Publisher Workflow [36] is an enterprise data publication platform that supports multiple data delivery types: via the download API, Zip archive, or WebDAV [72]. GO Publisher Workflow is ideal for exchanging large data volumes using open standards. Customers can download data using the Download API, fetching it directly from the database, or GO Publisher workflow can push it to third-party Content Management System.

4.6 An example: CoRED and GitHub

CoRED [75] is a web editor where multiple people can edit code at the same time. It is available as open source under Apache 2.0 license. CoRED is developed at Tampere University of Technology. The Source Code of CoRED [76] is available in GitHub. It is possible to either clone the Git repository with HTTPS or Subversion, or download the source code as a zip-package. License text can be viewed from the main page and source code files including installation instructions can be viewed from cored-folder.

According to one of the developers Antti Nieminen, CoRED was published in GitHub because it is easy to use, well-known, and trusted platform. There has been no technical problems in publishing. Release hasn't got any attention or gathered outside participants, but this was not the target of the release. To gather attention and participants, it would require for example advertising and writing documentation, which would help getting started. These things haven't been done much in this project. Although the repository is public, it was meant to be used inside the project.

4.7 AVAA open-access publishing platform

AVAA [68] is presented here as one possible publishing place for open source data. AVAA is an open-access publishing platform where users, such as research groups can suggest their own web applications or web page presentations found on the server of their own organization for publication in the application gallery. The purpose of AVAA [69] is to: “offer a platform and web-based tools for different disciplines to enhance accessability of open data.” AVAA portal includes:

- Interfaces and download functionalities for distributing research data.
- Applications for analyzing and visualizing data.

Data that is published via AVAA [69] can be stored to IDA Storage Service [70], or on the data provider's or owner's own servers. Data and applications-page lists links to research data and applications. The current applications in AVAA [68] are from different disciplines, for example atmospheric sciences, geosciences, and language research. With these applications user can for example download atmospheric data, study spreading of pollution clouds on a map, or use OpenStreetMap-application trough interface service. For Geosciences OpenStreetMap-application [71], there is also a link for downloading a zip-package. There are also links to map-images in several formats including WMS(Web Map Services), WMTS(Web Map Tile Service), and PNG.

4.7.1 IDA Storage Service

IDA Storage Service [70] is meant for storing data and metadata. The service is intended for the projects of Finnish universities, universities of applied sciences and projects of Academy of Finland. Universities and institutions have their own IDA contact person, who anyone interested in using IDA should contact to. A user permit application must be submitted to the contact person. Once the institution accepts the application, it is signed by the IDA contact person and sent to the IT Center for Sciences (CSC) [73].

IDA is based on open source iRODS [74] technology, which supports following properties [70]:

- Data preservation: storage procedures, virus scans
- Data management: project group distribution, management of copies
- Reporting: project specific reporting on storage space used
- Haka user identification

iRODS is released under a BSD license.

IDA [70] is part of forthcoming long-term data preservation service and guarantees at least three petabytes in storage capacity. The data owners can decide on openness and usage policies for their own data. IDA can be used with different user interfaces, browsers, online directories and command lines. All data stored to IDA must be described in metadata. Metadata requirements include defining usage rights for the data. URL-link to the actual license text must be declared as well as URL-links to license contract document and usage rights description document.

4.7.2 Comparing AVAA and IDA with data management systems

Compared to other data management systems introduced in section 4.5, IDA has clear instructions for applying user permits. Whereas DSpace is suitable for commercial organizations, AVAA is more directed for the use of research organizations, like universities. CKAN's [34] JSON api is used for adding and editing data, and in AVAA the numerical data can be retrieved in JSON-format. AVAA and CKAN are both aimed at data publishers wanting to make their data open and available. They also offer tools to analyze and visualize data, as well as license information. If the data that is published via AVAA is stored via IDA, then there is the same ability to store metadata that is also available in DSpace and CKAN.

4.8 Proposal for license and publishing platform

License for HTML5 agent framework should be a copyleft license like GPL-3.0 or AGPL-3.0. These licenses ensure that the software stays open, and other developers can make improvements to the framework and develop new applications. Licenses are also well known, high-quality, and revisions from previous versions of themselves. For publishing place, GitHub would be suitable, if there is also need to host a project where framework is further developed. GitHub is very popular with over eight million users and over nineteen million repositories [77]. If there is no need for project hosting, CKAN offers good features for publishing, finding, storing and managing data.

5 PRODUCTIZATION

5.1 Introduction

According to [26], **productization** means making software work for the general case and making it as easy as possible to use. According to [25] productization is a process where software is converted for re-use into a product. The target of productization is configurable application built upon a programming framework. Productized code is typically easy to install, configure and use. Code is made using coding, file format and interface standards. According to [11], productization includes technological elements from designing a product to commercial elements of selling and distributing a product. Products vary from standard “packaged” software products to customer tailored software. Examples of packaged software products are word processing packages, spreadsheets, business software, and operating systems.

Productization tends to came late to open source products. In many projects attitude towards productization is dismissive. This weakens project's appeal to beginner and intermediate skill levels in IT departments and other groups. Productization can help making software available to a larger community. It can reduce the learning curve for project participants. Lack of productization causes the company support costs or customer frustration. [26]

5.2 Installation

Work that takes place during productization includes writing installation scripts, creating graphical configuration tools, and creating engineering and end user documentation. **Installation** is the process of fitting software into the environment in which it will run. For an open source product, installation might mean unzipping the source code, compiling the program, and then figuring out how to fit it into your production or development environment. [26]

According to [27] installation precondition verification includes:

- Making sure that you have enough free space for the target application.
- “If your software requires any connections to other entities, check them *before you* begin the actual installation.”

Things to consider during the installation process [27]:

- Tell the user how much time the installation process will take.
- Provide the user information about installation progress. This can be provided with the progress meter or via command-line or user interface.
- Record the operations performed and the actions taken during installation to the log files. Capture user responses to questions raised by the installation.
- Make the installation process interruptible, with the expect that it will be interrupted at least once.
- Or make the installation process self-aware, so that it can restart itself if needed.
- Other option is to provide an installation checklist that breaks up installation process into series of smaller steps.
- Follow platform specific guidelines.
- Avoid asking unnecessary questions.

Postinstallation confirmation [27]

- Verify that the right files were copied to right locations, or execute the installed software and invoke any number of manual and/or automated tests. Check if the system is giving the right results. Log the results of postinstallation confirmation.
- Test installation and uninstallation.
- Provide automation scripts for your setup.

5.3 Configuration

Configuration [26] is the process of controlling the software's behavior. Mechanisms controlling software behavior vary from settings in property files and databases to small scripts that are executed at predefined points. Mechanisms can be undocumented, or documentation can be in the form of comments included in property files. The best programs have interactive environments for changing settings. For open source software, mechanisms are found more trough trial and error or reading code, rather than trough some comprehensive documentation.

Beginners need information about what settings are needed to make databases, web-servers, and network ports to behave properly. For beginner and intermediate level users the most useful documentation comprises a step-by-step tutorial of how to get a project running.

Configuration principles according to [27]:

- Something should be a configuration parameter only if the system can't function without it being set (such as a proxy server). Otherwise, make it a customization parameter and provide a reasonable default.

- Entities (a human, such as a system administrator; another system; or the system itself) setting the configuration values need "to know what values to set, how to set them (including formatting and valid values), why to set them, and their effect on system operation." This information should be given in the configuration file, not in external document.

Configuration parameter heuristics according to [27]:

- Make the configuration parameters easy to change. Store them externally, in a simple, easily managed nonbinary format.
- "Store all of the data in one location."
- "Make configuration data easy to capture and forward to technical support."
- "Make it hard to get the values wrong. If they are wrong, notify the user as soon as possible and either stop execution or continue with sensible defaults." Gather log data while the system is running and offer it to the user. Give the user a possibility to turn log data on and off while the system is running.

System needs to be configured before it begins execution, and during its operation. It needs to be clear, is the configuration parameter used only during initialization, or can it be changed while the system is running.

5.4 Software productization process

Product software is defined in [14] as "a packaged configuration of software components or a software-based service, with auxiliary materials, which is released for and traded in a specific market." In [12] productization is defined as a transformation process from developing customer specific software to product software. Software productization process introduced in [12] consists of six different stages (see figure 5.1).

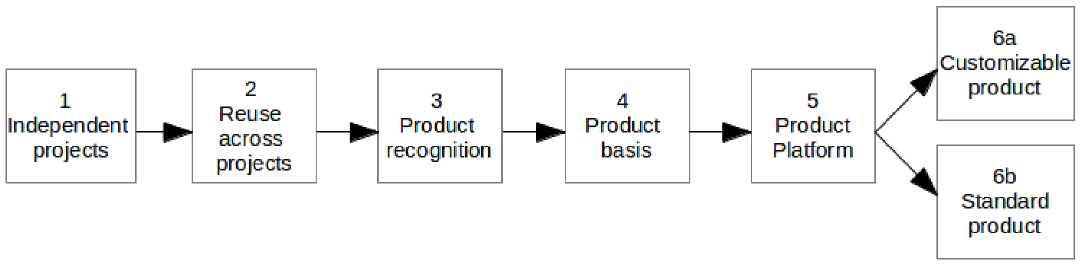


***Figure 5.1**: Software productization process [12]*

In stage 1 company has independent customer driven projects producing software applications.

In stage 2 features or functionalities are reused across projects. Projects use standardized features, but the amount of developed customized features is still larger.

In stage 3 the standardized part of the projects is larger than the customized part. This stage is called product recognition, first step of creating a product. Storing and managing customer requirements in one central place is important, because satisfying customers is the main focus of this stage.

In stage 4 product is recognized and product platform is used as a basis. The focus of satisfying the customers is decreasing and the focus of gaining more market share is increasing.

Stage 5 is called standardizing product platform: “increasing the set of features that form a common structure and introduce releases, from which still a stream of derivate products can be efficiently customized, developed and produced.” Focus starts to change from customer orientation to market orientation. Customer specific parts can be created if necessary.

In stage 6a there is a need for customizable part on product. Compared to stage 5, releases are the same for each customer.

Stage 6b is defined as standard software product. No customized features are included in product and product is completely configurable. [12]

6 INSTALLATION GUIDE

This installation guide is made for Ubuntu 12.04 LTS. Guide is valid for platforms with 64-bit operating system, 2 Gigabytes of memory, and AMD Turion 64 X2 Mobile Technology-processor. Installation guide was made with a laptop computer.

6.1 Unpacking files

Download *LakeusSrc.tar.gz*-package and unzip it to new folder. In this example *LakeusSrc*-package is downloaded to the home folder (/home/*username*).

Open terminal window and check (with *ls -l*) that you have *LakeusSrc.tar.gz*-package in your home folder. Extract tar.gz-package:

```
$ tar -zxvf LakeusSrc.tar.gz
```

After this command you should have a folder named *LakeusSrc* in your home folder, including all agent platform files.

6.2 Preparations for testing the installation

To test the installation of the agent framework with an example gmonitor application on a local server, the agent must be configured. Configuring the agent is done by modifying configuration.js-file, which includes URLs to:

- The origin server of the agent,
- list view of executing agents in server,
- agent server, where agent is uploaded. [3]

An example gmonitor application also needs URL to server where agent gets load average data. Function that returns this URL is called *loadavUrl* in the configuration-file.

The origin server of the agent is responsible for maintaining and serving all the agent platform files. Agent server receives the description of the agent in an HTTP POST request [6]. In the list view, a list of active agents running in the agent server is shown. List shows the id, URL, and running status of agents.

6.3 Fitting agent platform to own environment

Go to folder where you unzipped *LakeusSrc*-package and open configuration.js-file. Modify the return values shown in listing 6.1. You can change different host and port for each URL (an example is shown in listing 6.1 and concrete values for this example in listing 6.2.) In this example the new hosts are named *myhost1*, *myhost2*, and *myhost3*. New ports are *port1*, *port2*, and *port3*. In test configuration all servers are on the same host, so all hosts are named as *localhost*. If your servers are running on different devices, you have to make different configuration for each device (in this case the port numbers can be the same on each configuration). You can modify the return values of the following functions:

- *listUrl*: Returns URL to list view of executing agents in server. Host and port number are the same as in uploadUrl. /list (HTTP GET) gets a list of active agents as an HTML-file that can be shown in browser [5].
- *uploadUrl*: Returns URL to server where agent will be uploaded. The server can be running on different device than the origin server of the agent. In this case host part must be modified. /upload (HTTP POST) sends URLs to agent code and user interface together with serialized state [5].
- *applicationUrl*: Returns URL to origin server of the agent. Knows the location of the JavaScript and HTML-files needed to run the agent application.
- *loadavUrl*: Returns URL to server where agent starts asking load average data. If the server is on different device, host must be modified. This URL is not part of the agent platform, and is modified because an example gmonitor application uses it. [3]

More information about these URLs can be found from section 3.3. Script for changing these URLs from the command line is described in section 6.9. Modifications that help to change the ports of agent-, file-, and load-servers have been described in section 6.10.

Listing 6.1: Highlighting the modifiable parts of URL.

LakeusSrc/configuration.js

```
this.listUrl = function () {
      return "http://myhost2:port2/list";
}
this.uploadUrl = function () {
      return "http://myhost2:port2/upload";
}
this.loadavUrl = function () {
      return "http://myhost3:port3";
}
this.applicationUrl = function () {
      return "http://myhost1:port1";
}
```

Listing 6.2: Concrete example of modified values:

LakeusSrc/configuration.js

```
this.listUrl = function () {
      return "http://localhost:8891/list";
}
this.uploadUrl = function () {
      return "http://localhost:8891/upload";
}
this.loadavUrl = function () {
      return "http://localhost:8892";
}
this.applicationUrl = function () {
      return "http://localhost:8890";
}
```

If you want to test installation with an example gmonitor application, edit script src-value in gmonitor.html. Modifiable parts are shown in listing 6.3 and concrete example values in listing 6.4.

Listing 6.3: Highlighting the modifiable parts of URL:

LakeusSrc/gmonitor.html

```
<script src="http://myhost2:port2/configuration.js"></script>
```

Listing 6.4: Concrete example value:

LakeusSrc/gmonitor.html

```
<script src="http://localhost:8890/configuration.js"></script>
```

Change the original host and port of the URL so that they match the host and port of your application server. This ensures that the gmonitor application running on browser can get the configuration.js-file.

6.4 Node.js installation

Because servers have been implemented with node.js, it must be installed before servers can be started.

Open new terminal window. Easiest way to open a new terminal window is to press *Ctrl + Alt + T*. To install nodejs manually, do following steps [4]:

1. Install nodejs. It is needed to run agent-, load-, and file-servers:

```
$ sudo apt-get install nodejs
```

2. Install curl, which is needed to get npm installation script from npmjs website:

```
$ sudo apt-get install curl
```

3. Install Node package manager, which is needed for installing mime- and socket.io-packages. To install npm, run script *npminst.sh* from *lakeus*-folder.

```
$ sudo ./npminst.sh
```

Optionally, you can run installer from npmjs web page:

```
$ curl https://www.npmjs.org/install.sh | sudo sh
```

4. Install mime-module to nodejs:

```
$ sudo npm install mime
```

5. Current agent server also needs socket.io module for messaging with client:

```
$ sudo npm install socket.io
```

6.5 Starting the servers manually

If you want to start servers manually, open new terminal window and move to *LakeusSrc*-folder. Do following steps:

1. Start fileserver:

```
$ node fileserver.js
```

2. Open new terminal window and move to *LakeusSrc*-folder. Start loadserver:

```
$ node loadserver.js
```

3. Open new terminal window and move to *LakeusSrc*-folder. Start agentserver:

```
$ node agentserverindex.js
```

Now you should have three terminal windows open, one server running on each.

6.6 Starting the servers by using a script file

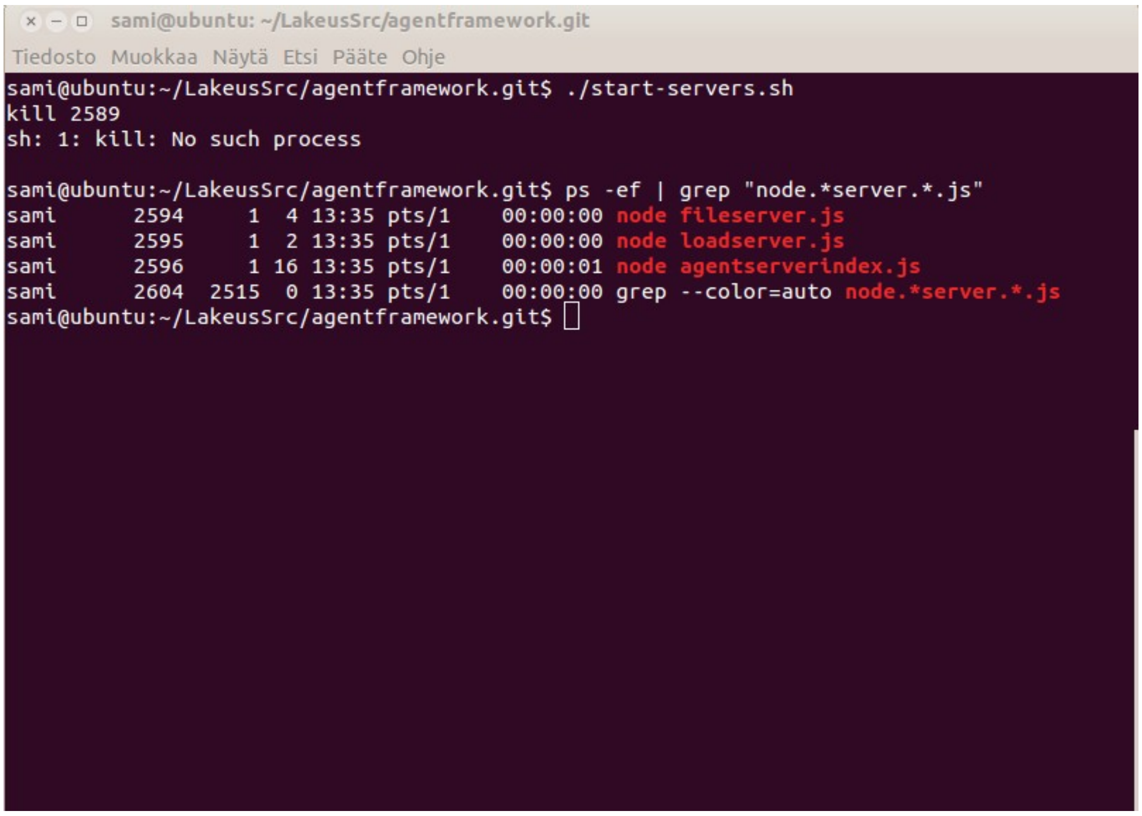
Open new terminal window. You can start servers by running the script:

```
$ ./start-servers.sh
```

You can check if the servers are running:

```
$ ps -ef | grep "node.*server.*.js"
```

Figure 6.1 shows command-line output when servers are started, and started servers are listed. The log-files of servers can be found from *LakeusSrc*/*logs*-folder. Log-files are also in html-form: Go to *logs*-folder and open file *index.html* with your browser. You should see a page with links to log-files. Press wanted log-file name to see the contents. You can also see the log-files by pointing your browser to *applicationURL/logs/index.html.*


sami@ubuntu: ~/LakeusSrc/agentframework.git
Tiedosto Muokkaa Näytä Etsi Pääte Ohje

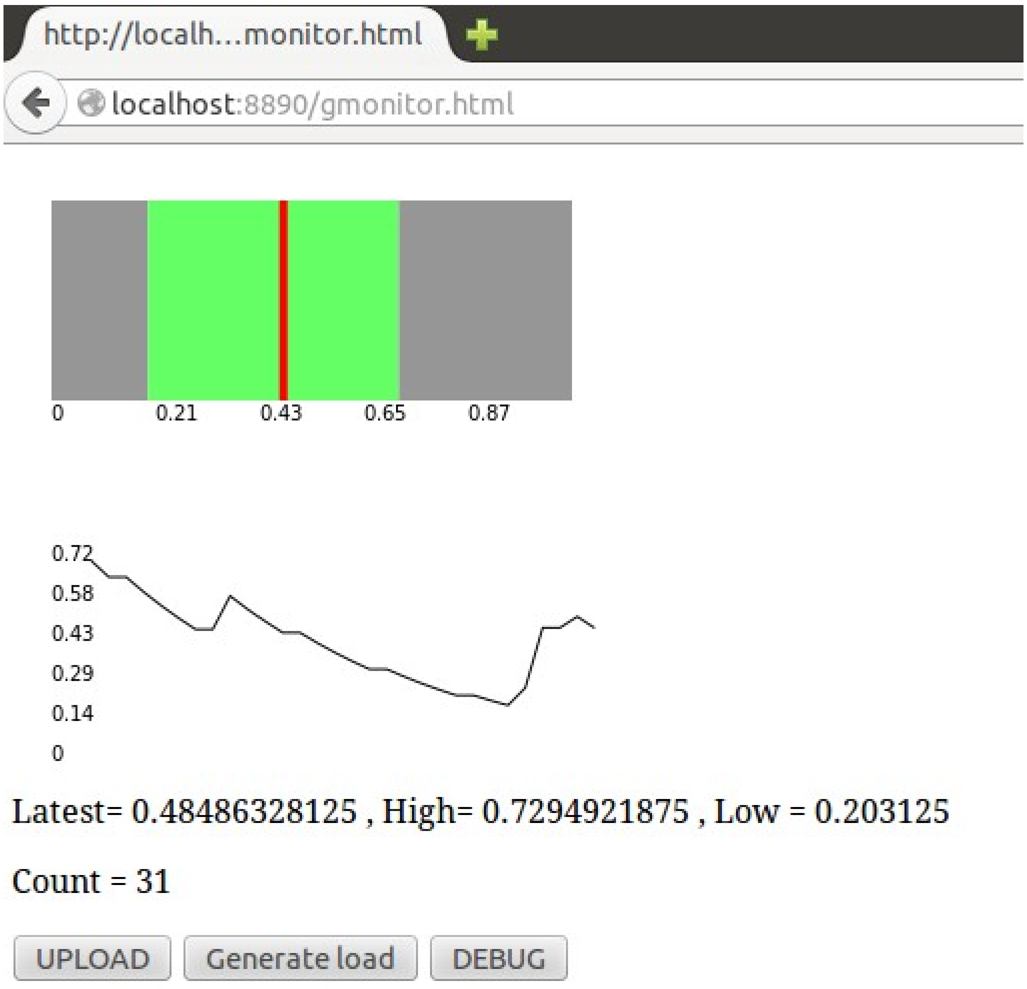
```
sami@ubuntu:~/LakeusSrc/agentframework.git$ ./start-servers.sh
kill 2589
sh: 1: kill: No such process

sami@ubuntu:~/LakeusSrc/agentframework.git$ ps -ef | grep "node.*server.*.js"
sami      2594     1  4 13:35 pts/1    00:00:00 node fileserver.js
sami      2595     1  2 13:35 pts/1    00:00:00 node loadserver.js
sami      2596     1 16 13:35 pts/1    00:00:01 node agentserverindex.js
sami      2604  2515  0 13:35 pts/1    00:00:00 grep --color=auto node.*server.*.js
sami@ubuntu:~/LakeusSrc/agentframework.git$ 
```

Figure 6.1*: Starting servers and checking that servers are running*

You should have three servers running: *fileserver.js*, *loadserver.js*, and *agentserverindex.js*. If a server is missing from listing you should check error output from a server log-file.

6.7 Testing the installation with gmonitor

To test installation of the agent framework with an example gmonitor application (figure 6.2), point your browser to *applicationURL + "/gmonitor.html"*. In this example it is *http://localhost:8890/gmonitor.html*.



Latest= 0.48486328125 , High= 0.7294921875 , Low = 0.203125

Count = 31

***Figure 6.2**: gmonitor application*

Gmonitor starts asking load average data from the loadavUrl that you have defined earlier. In this example the loadavUrl is *http://localhost:8892*. When you press *upload*-button, the state of the agent is serialized and agent is transferred to the agent server. Listing 6.5 shows the example of an agent description from the agent_log-file.

Listing 6.5: Example agent description.

LakeusSrc/logs/agent_log_xxxx

```
{"auri": "http://localhost:8890/gmonitor.js",
 "huri": "http://localhost:8890/gmonitor.html",
 "id": "241594",
 "curi": "",
 "variables": { },
 "memory": {
```

```
 "high": 0.98583984375,
 "low": 0.833984375,
 "count": 4,
 "history": [0.984375,0.98583984375,0.90673828125,0.833984375]}
}
```

An agent description contains the following information [6]:

- *auri*: “a URL that points to the JavaScript-file of the application.”
- *huri*: a URL that points to HTML-file, the user interface of the application.
- *id*: “unique ID of the agent instance.”
- *variables/memory*: “local state in terms of names and values of local variables.”

In the user interface, a list of active agents is shown. List shows the id, URL, and running status of agents.

6.8 Automatizing Node.js installation

Node.js installation script does the steps 1-5 defined in chapter 6.4. It installs node.js and uses npm to install mime- and socket.io-modules. Script is shown in listing 6.6.

Listing 6.6: nodejs_install script.

LakeusSrc/nodejs_install.sh

```
#!/bin/sh

sudo apt-get install -y nodejs
sudo apt-get install -y curl
sudo ./npminst.sh
sudo npm install mime
sudo npm install socket.io
```

Script can be started from terminal window with *./nodejs_install.sh*-command.

6.9 Automatizing the changing of URLs

change_urls.sh (listing 6.7), is a shell-script for changing configuration URLs to *configuration.js*-file from command-line. Script can be started from terminal window with *bash change_urls.sh*-command. Script automates changing configuration-file URLs that was made in chapter 6.3 by changing configuration-file directly in editor.

Listing 6.7: change_urls.sh

LakeusSrc/change_urls.sh

```
#!/bin/sh

# Copy configuration.js to backup-file
cp configuration.js configuration_backup.js

# Get all URLs from configuration-file
array=( $(grep http ./configuration.js | grep -Eo '["].*["]'))

# Get all URL-names from configuration-file
array2=( $(grep this. ./configuration.js | grep -Eo '[.].*[l]'
| tr -d .))

# Splitting array to get only url names
array2=("${array2[@]:0:6}")
#printf "%s\n" "${array2[@]}"
declare -i j=0

for i in "${array[@]}"
do
  url=$i
  # Saving default URL to new_url-variable
  new_url="$url"
  name="${array2[$j]}"
  # Printing the name of url
  printf "\nSet new $name:\n"
  # Print new line with default url
  read -i "$url" -e newline

  #If new line is not empty, new line is saved to new_url-
variable
  if [ -n "$newline" ]
    then new_url="$newline"
  fi

  # Replace URL with new URL in configuration.js-file
  sed -e '/'$name'/{ N; s#'$url'#'$new_url'#g }'
"configuration.js" > "configuration.js.tmp" &&
mv "configuration.js.tmp" "configuration.js"
```

```
j=$((j+1))
done
```

Configuration-file is stored to backup-file (configuration_backup.js) so that user can restore it if something goes wrong when setting new URLs. Script finds URLs and URL-names from configuration.js file using *grep*. Script stores URLs and URL-names to arrays. Script goes url-array trough in for-loop and asks user to set new URL. New URL is changed to configuration-file using *sed*. If user enters empty line, the old URL is stored to configuration-file. *Sed* uses tmp-file to restore changes if changing URL causes an error.

6.10 Removing hard-coded server port number-values

6.10.1 parsePort-function

As mentioned earlier in section 3.3, there were hard-coded port number-values in the *fileserver.js*-, *agentserver.js*-, and *loadserver.js*-files. Problem arises in a situation, where user changes port number of certain configuration URL to configuration file, but doesn't change port number to server file. If user changes applicationURL port, the port number should be changed to *fileserver.js*-file as well. Otherwise application wouldn't start in browser. To solve this kind of problem, modifications to the configuration-file have been done to get the port numbers directly from configuration URLs. New *parsePort*-function (listing 6.8) is added to configuration-file for parsing agent-, file-, and load-server ports from URLs.

Listing 6.8: Function for parsing port from URL.

LakeusSrc/configuration.js

```
this.parsePort = function (url) {
     var arr1 = url().split("/");
     var arr2 = arr1[2].split(":");
     return arr2[1];
}
```

parsePort-function splits URL into parts using “/”-marks as a delimiter. Split parts are stored to *arr1*-array. The part of the URL including host and port divided by “:”-mark is then split and stored to *arr2*-array, and second item of array is returned.

6.10.2 serverPort-functions

Agent-, file-, and load-servers are calling different function to get the port-value. These functions are *agentserverPort*, *fileserverPort*, and *loadserverPort* (listing 6.9).

Listing 6.9 Functions returning server ports.

LakeusSrc/configuration.js

```
this.agentserverPort = function () {
      return this.parsePort(this.uploadUrl);
}

this.fileserverPort = function () {
      return this.parsePort(this.applicationUrl);
}

this.loadserverPort = function () {
      return this.parsePort(this.loadavUrl);
}
```

These functions call *parsePort*-function for getting the port from right URL and return the result of *parsePort*-function to servers.

7 PROGRAMMING GUIDE

7.1 Generic agent class

Generic agent [3] is the base class for each agent application. Generic agent should not be instantiated, because its purpose is to provide the generic parts of the agent to all agent applications. Inheritance between the generic agent and the application agent is done by using *functional inheritance pattern* described in [32].

7.1.1 Functions

Generic agent is responsible for preserving the agent state. Saving inner state of agent is done using *registerVar(variable name)* -function. It registers variables to the state of the agent.

Another function for the *state management* is:

- *setMemory(variable list)*: Resets the memory of the agent to the given variable list. Clears all the previously saved memory variables and their data.

Functions for *serialization and transfer management*:

- *serialize()*: Returns agent description. This method is for the framework and not usually called from the application code.
- *preupload()*: Used when agent needs to do something before the upload. Generic agent has no implementation for preupload.
- *upload()*: Calls *serialize*-function and transfers the agent description to the server.

Functions for the *execution management* (of the agent) are:

- *continueWork()*: Initializes the arriving agent and returns the agent variables and (memory) in restart of the agent.
- *setRunInterval(interval value)*: Sets the interval in milliseconds for the calls of *work()*-function.
- *work()*: Defines what agent is supposed to do (in defined intervals). Generic agent has no implementation of work-function.

- *setWork(name of work-function):* Sets the defined *work*-function for the agent. Enables the use of other function name than *work*. Changing *work*-function during the agent execution is possible.
- *start()*: Starts agent.
- *stop()*: Stops agent execution.
- *getRunning Status()*: Gives the status information of agent. Used when agent is in server.
- *isInServer()*: Returns true if agent is in server, otherwise returns false.

[3]

7.1.2 Variables

In order to get the agent state private, variables and functions are assigned to members of *that* [32]. *that* is a variable where all the functions and variables that application agent has access should be saved.

- *src* contains URL to JavaScript-file, functionality of the application. Src can be an array of URLs to the JavaScript-files, if the agent consists of multiple files.
- *html* contains URL to the html-file, user interface of the application, which has references to CSS-files and other needed files, like image files. [3]

7.2 Application agent

Application agent [3] is a concrete implementation of an application that can travel between browser and server. Application agent's variable *that* should include all variables that are saved to memory so generic agent can find them.

7.2.1 Compulsory functions for application agent

- *work()*: Things agent needs to do to update it's state, for example query data. (If *work()* -function should be changed, agent must be stopped before *setWork()* -call and started after *setWork()* is called.)
- *continueWork()*: Initializes or resets application specific parts that are not saved in memory or variable list of an agent. (E.g. Values of drop down menus in browser). Also *setRunInterval()*- and *start()*-functions of the generic agent should be called in this function. [3]

7.2.2 Additional compulsory functions for application agent

- *CreateAgentObject(location of the javascript and html-files of the agent)* is a factory method that creates agent application object without including application specific code to the agent server. This function is needed in both browser and server.

- *InitExecution()* is called in HTML-body, agent is initialized and started before UI is shown to the user using *onLoad()*-function. [3]

7.3 Creating an example agent application

When starting to create an application agent, general guidelines introduced in [3] section 5.1 should be followed. This tutorial explains the steps of creating a simple agent application that increases index on each call of *work*-function. Functionality of the application is written in *index_count.js*-file and user interface of the application in *index_count.html*-file.

Listing 7.1 shows the first steps in creating a constructor: Variable *that* and a new instance of *Agent*-class are created. Application variable *index* is initialized and saved to the memory of the agent with *registerVar*-function.

Listing 7.1: Initializing and registering the agent variables.

LakeusSrc/index_count.js

```
function Count(src,html){

     var that = new Agent(src,html);

     that.index = 0;
     that.registerVar('index');
```

In second step (listing 7.2) application specific *continueWork*-function is extented to variable *that.continueWork* is used for initializing the agent. Interval for the calls of *work*-function is set in *restartWork*-function.

Listing 7.2: Extending continue work-function.

LakeusSrc/index_count.js

```
var agentContinueWork = that.continueWork;

that.restartWork = function(){
     this.setRunInterval(4000);
}
```

Function *work* (listing 7.3) for increasing the *index* is extended to variable *that*. In *work*-function *index* is increased and printed.

Listing 7.3: work-function.

LakeusSrc/index_count.js

```
that.work = function(){
    that.index = that.index + 1;
    document.getElementById('index').innerHTML = that.index;
    console.log('Index: ' + that.index);
};
```

Application uses the same *upload*-function that is defined in *Agent.js*-file. *Upload*-function is used for serializing agent state and sending it to server. Variable *that* is returned in the end of constructor.

After creating a constructor, separate *createAgentObject*-function is created for creating new Count-agent. Function is showed in listing 7.4.

Listing 7.4: Creating new Count-agent.

LakeusSrc/index_count.js

```
function createAgentObject(src, html){
    return new Count(src, html);
}
```

initExecution-function is created for initializing agent execution. Function creates src- and html-variables and calls *createAgentObject* and *continueWork*-functions. *InitExecution*-function is showed in listing 7.5.

Listing 7.5: Initializing agent execution.

LakeusSrc/index_count.js

```
  function initExecution(){
      var src = configuration.applicationUrl() +
"/index_count.js";
      var html = configuration.applicationUrl() +
"/index_count.html";
      agent = createAgentObject(src, html);
      agent.continueWork();
  }
```

Next step is to create index_count.html-file for the user interface of the application. Application is shown in figure 7.1.

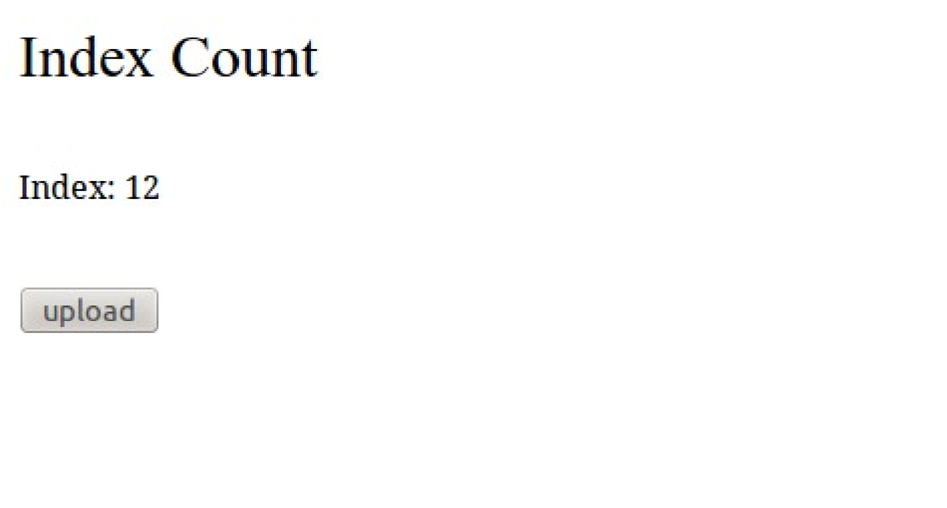


***Figure 7.1**: Index count*

index_count.html-file is shown in listing 7.6.

Listing 7.6: index_count.html.

LakeusSrc/index_count.html

```
<!DOCTYPE html>
<html>
<head>
     <title>Index count</title>
     <meta name="viewport" content="width=device-width">

     <script
src="http://localhost:8890/configuration.js"></script>
     <script>
          configuration.downloadAgent("index_count.js")
     </script>
     <style> h1{font:30px Times New Roman}</style>
</head>

<body onload=initExecution()>
```

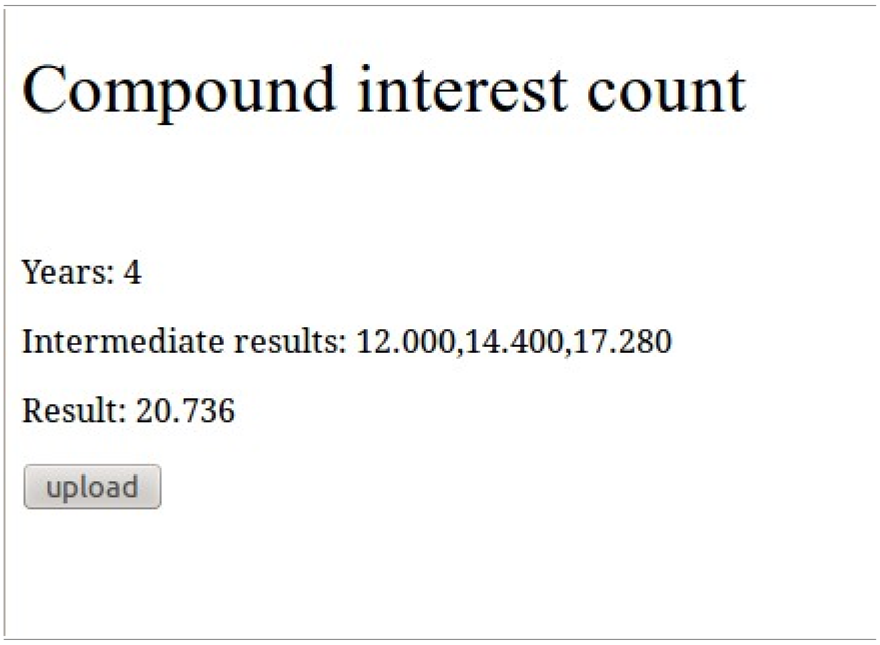
```
        <h1>Index Count</h1>
        <div style= height:20px></div>
        Index:
        <span id='index' value="index"></span>
        <div style= height:40px></div>
        <input type='button'
        onclick="agent.upload()"
        value="upload">
</body>
</html>
```

In the head tag, *configuration.js*-file is first downloaded from application server, and then *index_count.js*-file is downloaded using *downloadAgent*-function in the configuration-file. Agent execution is started by calling *initExecution*-function. When user presses upload-button, the *upload*-function of the agent is called and agent is sent to the agent server.

7.4 Adding new values and functions to Count-agent

Second example is build using *index_count.js*-file as a basis of both agent applications. Purpose of this example is to show how agent-to-agent communication works. Application that counts compound interest is created (see figure 7.2). It multiplies the starting value with a certain multiplier and continues to multiply the result value of previous count on each execution of work-function. Counting is done in phases, because intermediate results are stored and shown to user.

In section 7.5, communication component is added to counting application for sending the counting results to receiver agent, which receives the counting results and shows them to user. Application codes are written in *count.js*- and *receiver.js*-files, and user interfaces in *count.html*- and *receiver.html*-files.



Compound interest count

Years: 4

Intermediate results: 12.000,14.400,17.280

Result: 20.736

***Figure** 7.2: Compound interest count user interface*

New values are added to Count-agent constructor (see listing 7.7):

- Starting value for count,
- multiplier,
- result,
- results-array for storing several intermediate results is created,
- index for count (number of years).

Listing 7.7: New values in Count-agent constructor.

LakeusSrc/count.js

```
that.startValue = 10;
that.multiplier = 1.2;
that.result = 0.0;
that.results = new Array();
that.index = 0;

that.registerVar('startValue');
that.registerVar('multiplier');
that.registerVar('result');
that.registerVar('results');
that.registerVar('index');
```

Counting is done by calling *multiply*-function from *work*-function. In *multiply*-function (listing 7.8) provisional result is multiplied with multiplier and result is stored to *result*-variable and *results*-array. Index is increased by one.

Listing 7.8: Multiply-function.

LakeusSrc/count.js

```
// Does the counting and stores results
that.multiply = function(){
   console.log("in multiply");
   value = that.result;
   if(value === 0){
     value = that.startValue;
   }
   result = parseFloat(value) * parseFloat(that.multiplier);
   // result is the next value to be multiplied
   that.result = result.toFixed(3);
   that.results.push(result.toFixed(3));
   that.index = that.index + 1;
};
```

Work-function (listing 7.9) calls *multiply*-fuction for counting, *updateView*-function for showing results in browser, and *sendMessage*-function to send result-values.

Listing 7.9: Work-function for Count-agent.

LakeusSrc/count.js

```
// work-function that calls multiply for counting
// and sendMessage for sending the results.
that.work = function(){
   that.multiply();

   // Show results in browser
   that.updateView();

   // For server to show results
   console.log(that.result);
   console.log(that.results.slice(0,that.results.length-1));

   // Sends message to another agent
   that.sendMessage();
```

```
  };
```

When the agent is uploaded to server, counting index is shown as a running status.

7.5 Getting the communication working

Currently it seems that the working version of socket.io npm package is 0.9.4.
Open terminal window. Install socket.io and socket.io-client:

```
$ sudo npm install socket.io@0.9.4
$ sudo npm install socket.io-client@0.9.4
```

When the agent server is started from terminal, text “socket.io started” should be shown. CommunicationUrl in *configuration.js*-file should be set as "http://**myhost**:**myport**", where *myhost* is the hostname of your agentserver and *myport* is the port number of your agentserver.

In this example the communicationUrl is "http://**localhost**:**8891**". CommunicationUrl must be configured to ensure that the communication component communicates with the agent server. Communication component should be required at the beginning of *agentserver_handlers.js*-file. The beginning of the file should include the line:

```
var CommComponent = require("./communication.js").CommComponent;
```

Without requiring the communication component, uploading agent won't work.

7.5.1 Sending information from agent to another agent

Information sending from agent to another agent is made using the communication component. At first, communication component is required from *communication.js*-file at the beginning of *count.js*-file. Then a new *CommComponent*-variable is initialized in Count-agent constructor (listing 7.10). Callback-function is given as a parameter, so that communication component can provide the received information to Count-agent. Namespace where communication component will be connected is set as *“count”* by calling the *setNameSpace*-function. After that connection to the agent server is initialized and communication namespace is created by calling the *initIO*-function.

Listing 7.10: Creating and initializing communication component.

LakeusSrc/count.js

```
    that.commComponent = new CommComponent(function(message) {
      //alert(message);
  });
  that.commComponent.setNameSpace("count");
  that.commComponent.initIO();
```

Counting parameters and results are send to receiver agent by calling *sendMessage*-function of communication component (listing 7.11). One array for all values is created, and another array for counting parameters. Then counting parameters are added to *parameters*-array. *Parameters*- and *results*-arrays are added to *allData*-array, which is passed as a parameter to *sendMessage*-function.

Listing 7.11: Calling sendMessage-function.

LakeusSrc/count.js

```
  that.sendMessage = function(){
      var allData = new Array();
      var parameters = new Array();

      parameters.push(that.startValue);
      parameters.push(that.multiplier);
      parameters.push(that.index);

      allData.push(parameters);
      allData.push(that.results);

      that.commComponent.sendMessage(allData);
  };
```

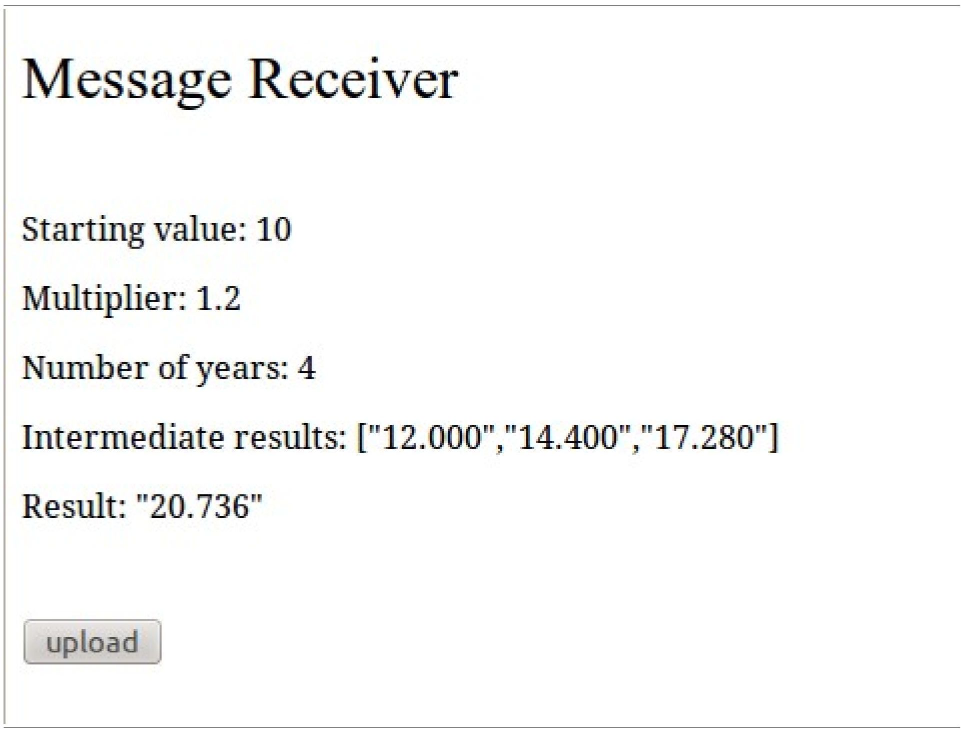
In *count.html*-file, *socket.io* from agent server must be included, because communication component requires *socket.io-client* to be able to work. Communication component from file *communication.js* must also be included. Files are included as *script src*-values:

```
<script src="http://localhost:8891/socket.io/socket.io.js"></script>
<script src="http://localhost:8890/communication.js"></script>
```

These lines are located between the <*head*>-tags in the *count.html*-file.

7.5.2 Receiving information from Count-agent

New Receiver-agent (figure 7.3) is created for receiving results from Count-agent.



Message Receiver

Starting value: 10

Multiplier: 1.2

Number of years: 4

Intermediate results: ["12.000","14.400","17.280"]

Result: "20.736"

***Figure** 7.3: Message receiver user interface*

When creating new communication component (listing 7.12), callback-function is given as a parameter. Inside callback function the received message is stored to local *msg*-variable and shown in console and user interface. When receiver-agent is at server, received result is shown as a running status.

Listing 7.12: Creating communication component

LakeusSrc/receiver.js

```
that.commComponent = new CommComponent(function(message){
    that.msg = message;
    console.log(message);
    that.updateView();
});
that.commComponent.setNameSpace("count");
that.commComponent.initIO();
```

Setting namespace and initializing connection are done same way as in Count-agent constructor. Namespace must be the same as the namespace set to Count agent, so it is set to *“count”*.

8 EVALUATION

This chapter contains an evaluation of installation and programming guides based on installation and configuration principles and general user manual and technical writing guidelines. According to [27] the most useful form of documentation for beginner and intermediate level users is a step-by-step tutorial of how to get a project running. This principle has been followed in chapter 6 and 7 by writing them in tutorial form.

Things that entities (a human, such as a system administrator; another system; or the system itself) setting the configuration values must know are [27]:

- What values they can set,
- how they can set the values,
- why they have to set the values,
- and how the values effect on system operation.

Installation guide's section 6.3 explains user what kind of URLs there are and what is their purpose. Then an example is shown where the configuration URLs are changed directly to the configuration-file.

From the installation principles described in section 5.2, only the one that advices to provide information about installation process to the user realizes, because node.js installation informs user what is being done by printing information to the terminal window. From the postinstallation configuration principles, the one that tells to execute installed software realizes, because in the installation guide the user is advised to test the installation of the agent framework with an example gmonitor application.

Following the configuration parameter heuristics described in chapter 5.2, the configuration parameters have been made easy to change by providing a configuration script for changing the URL-values from command-line. If user enters empty line, previous URL is stored to configuration-file. This configuration script is only meant to be used before the system is running. Scripts that enable configuration changes while the system is running could also be made. Graphical user interface for changing configuration, or ability to easily change configuration files by defining agent manifest files mentioned in [3] would be even better than the command-line based configuration.

In [42] a list of techniques is presented that might help writers to engage their audience. One advice for the writer is to “Be sure the sense of a paragraph does not rely on its title.” This advice is important, because some of the readers might skip over subtitles, headings and introductions. In this thesis, the first paragraphs of the installation and programming guides explain the mission of the chapter, so chapters are not relying on their title.

Ganier [43] gives design principles for improving procedural instructions. One of them is to use clear, precise and prominent headings to facilitate comprehension and execution of the instructions. In installation and programming guides headings are mostly named by the task or subject (class, filename) which is explained in the chapter.

Execution of the action is easier if the presentation order corresponds to the execution order [43]. The installation guide follows this advice by processing things in same order they should be executed. The execution steps should be numbered and presented in a vertical sequence to improve readability of the instructions and to allow users to find specific information more quickly [43]. In the installation guide, node.js installation instructions as well as instructions for starting the servers follow this advice.

In [44], using a conversation style and active verbs in user manuals is recommended to make the sentences simpler and shorter, define the responsibility of the action, and make the text more interesting. This advice has been followed in chapter 6 (in sections 6.1-6.7).

Using graphics, numbered or bulleted lists, charts, or tables instead of text is recommended where practical. Reason to use graphics, lists, charts, or tables is mentioned in [44]: “These consolidate related information in a logical format and reduce the amount of text a reader must digest.” In chapter 6, both numbered and bulleted lists have been used when describing the configuration URLs and explaining node.js installation steps. In chapter 7, functions and variables are presented as bulleted lists. Sommerville [45] advices that facts should be presented in a list rather than in a sentence.

Kennedy [46] advices to get to the point and avoid wordiness and repeating same things many times. Installation steps are explained in a very simple, short way in the installation guide. Installation guide introduces two ways for installing node.js and starting the servers. That's why user can skip some parts of the installation guide, and that's why some things like opening a new terminal window must be repeated many times.

9 CONCLUSIONS

This thesis provided basic information about mobile agent platforms and a short description about the agent platform used in this thesis. The term productization was explained, and especially installation and configuration principles during productization. A software productization process was also briefly described.

Main purpose of this work was to prepare a mobile agent platform for publishing by writing an installation and a programming guide for the platform. Open source licenses were studied, as well as open source software hosting facilities and data management systems. Productization in this thesis included installing the agent platform: Unpacking code, installing node.js and npm-modules, and configuring URL-values in configuration-file. These steps were then documented to the installation guide. Starting agent servers and installation testing with an example gmonitor application were explained. A script for node.js installation was made, as well as a script for changing URLs easily from command-line. Script-files are described in listing 9.1.

Listing 9.1: Script-files

Filename	**Purpose of file**
nodejs_install.sh	Installs node.js. Installs mime- and socket.io-modules to node.js.
change_urls.sh	User can modify configuration URLs from command line and changes are made directly to the configuration-file.

New functions were added to the configuration-file to offer port numbers of servers directly from the configuration-file instead of having hard-coded port-number values in each server file. Server-files were modified to call these functions. Communication component variable was included to *agentserver_handlers.js*-file. All modified files are listed and modifications are explained in listing 9.2.

Listing 9.2: Modified files

Filename	**Modifications**	**Purpose of modifications**
agentserver_handlers.js	Included communication component-variable.	Getting agent-to-agent communication working

		when one of the agents is uploaded to agent server.
configuration.js	Added new parsePort-function that parses agent-, file-, or load-server ports from configuration URLs.	Getting server port directly from configuration URL.
configuration.js	Added new serverPort -functions to get agent-, file-, or load-server ports from configuration-file.	Functions that agent-, file-, and load-servers can call to get the port-number from configuration URL.
agentserver.js	Calls agentserverPort-function of configuration-file to get the port directly from configuration URL.	Removed hard-coded agentserver port number. Agentserver port must be changed only to configuration-file.
fileserver.js	Calls fileserverPort-function of configuration-file to get the port directly from configuration URL.	Removed hard-coded fileserver port number. Fileserver port must be changed only to configuration-file.
loadserver.js	Calls loadserverPort of configuration-file to get the port number directly from configuration URL.	Remover hard-coded loadserver port number. Loadserver port must be changed only to configuration-file.

Classes and variables for generic and application agent were introduced in the programming guide. Examples for creating an agent application and using agent functions were presented. Communication component usage was described with an example, where agent sends counting results data to another agent. New files created for the examples are described in listing 9.3.

Listing 9.3: New code files

Filename	**Purpose of file**
index_count.js	Example agent application that counts index. Getting started with coding agent application with the agent platform.
index_count.html	User interface of index_count-application.
count.js	Example agent application that counts compound interest and sends counting results to receiver-agent. Trying agent-to-agent communication using communication

	component.
count.html	User interface of count-application.
receiver.js	Example agent application that receiver counting results from count-agent application and shows results in user interface.
receiver.html	User interface of receiver-application.

Installation and programming guides were evaluated based on installation and configuration principles and general user manual and technical writing guidelines.

REFERENCES

[1] Braun, P. Rossak, W. Mobile Agents: Basic Concepts, Mobility Models, and the Tracy Toolkit, 2005. Science Direct E-book, pp. 7-213.

[2] Mitrović, D., et al., Radigost: Interoperable web-based multi-agent platform. J. Syst. Software (2014), http://dx.doi.org/10.1016/j.jss.2013.12.029.

[3] Järvenpää, L. Development and evaluation of HTML5 agent framework. Master of Science Thesis.

[4] GitHub. Istalling Node.js via package manager. [WWW]. [Referred 28.5.2014]. Available: https://github.com/joyent/node/wiki/Installing-Node.js-via-package-manager.

[5] Järvenpää, L et al. Mobile Agents for the Internet of Things. IEEE, 2013.

[6] Systä, K., Mikkonen, T., Järvenpää, L. HTML5 Agents – Mobile Agents for the Web. Springer-Verlag Berlin Heidelberg, 2011.

[7] Borselius, N. Mobile Agent Security. Electronics & Communication Engineering Journal, October 2002.

[8] Aglets. [WWW]. [Referred 24.6.2014]. Available: http://aglets.sourceforge.net/.

[9] Wong, D et al. Java-Based Mobile Agents. Communications of the ACM, March 1999/Vol. 42, No. 3.

[10] Dömel, P. Mobile Telescript Agents and the Web. Compcon '96. Technologies for the Information Superhighway, Digest of Papers, 1996, pp. 52-57.

[11] Hietala, J., Kontio, J., Jokinen, J., Pyysiainen, J. Challenges of Software Product Companies: Results of a National Survey in Finland. 10th International Software Metrics Symposium, 2004, pp. 232–243.

[12] Artz, P., Van De Weerd, I., Brinkkemper, S., Fieggen, J. Productization: Transforming from Developing Customer-Specific Software to Product Software. Lecture Notes in Business Information Processing, 2010, Vol.51, pp. 90-102.

[13] Meyer, M.H., Seliger, R. Product platforms in software development. Sloan Management Review vol. 40, 1998, pp. 61–74.

[14] Xu, L., Brinkkemper, S. Concepts of product software. European Journal of Information Systems vol. 16, no. 5 (Oct 2007), pp. 531-541.

[15] SourceForge. [WWW]. [Referred 9.8.2014]. Available: http://sourceforge.net.

[16] Lauchpad. [WWW]. [Referred 9.8.2014]. Available: https://help.launchpad.net/Projects/FileDownloads#Publishing_files.

[17] Open Source Licenses. [WWW]. [Referred 10.8.2014]. Available: http://opensource.org/licenses.

[18] Choosing Open Source License. [WWW]. [Referred 11.8.2014]. Available: http://choosealicense.com/licenses/.

[19] GitHub help. Open Source Licensing. [WWW]. [Referred 11.8.2014]. Available: https://help.github.com/articles/open-source-licensing.

[20] GitHub help. Support for subversion clients. [WWW]. [Referred 11.8.2014]. Available: https://help.github.com/articles/support-for-subversion-clients.

[21] Launchpad software collaboration platform. [WWW]. [Referred 11.8.2014]. Available: https://launchpad.net/.

[22] Google Project Hosting. [WWW]. [Referred 12.8.2014]. Available: https://code.google.com/projecthosting/.

[23] BitBucket. [WWW]. [Referred 12.8.2014]. Available: https://bitbucket.org/features.

[24] Codehaus. [WWW]. [Referred 12.8.2014]. Available: http://www.codehaus.org/.

[25] YoLinux.com. Software Productization: Software Architecture and Design for Productization: Description of software architecture, design and concepts to support

software “productization”. [WWW]. [Referred 7.9.2014]. Available: http://www.yolinux.com/TUTORIALS/SoftwareArchitecture-Productization.html.

[26] Woods, D. Guliani, G. Open Source for the Enterprise: Managing Risks, Reaping Rewards. O'Reilly Media, 2005. p. 9, 21,72. pp: 29-44, 105-126.

[27] Hohmann, L. Beyond Software Architecture: Creating and Sustaining Winning Solutions. Addison Wesley Professional, 2003. pp. 203-234.

[28] An Overview of the Grasshopper Agent Platform. [WWW]. [Referred 24.9.2014]. Available:
http://cordis.europa.eu/infowin/acts/analysys/products/thematic/agents/ch4/ch4.htm#2.

[29] Node.js. [WWW]. [Referred 25.9.2014]. Available: http://nodejs.org/.

[30] Socket.io. [WWW]. [Referred 25.9.2014]. Available: http://socket.io/.

[31] Inversion of Control Containers and the Dependency Injection Pattern. [WWW]. [Referred 26.9.2014]. Available: http://martinfowler.com/articles/injection.html.

[32] Crockford, D. Javascript: The Good Parts. O'Reilly Media, Inc., 2008. 172 p.

[33] Carzaniga, A., Picco, G., P., Vigna, G., 1997. Designing distributed applications with mobile code paradigms. In Proceeding of the 19th international conference on Software engineering (ICSE'97), May 17-23, 1997, Boston, Massachusetts, USA. pp. 22-32.

[34] CKAN – The open source data portal software. [WWW]. [Referred 2.10.2014] Available: http://ckan.org/.

[35] CKAN Docs: API Guide. [WWW]. [Referred 2.10.2014]. Available: http://docs.ckan.org/en/latest/api/index.html.

[36] Snowlake Software: Go Publisher Workflow. [WWW]. [Referred 2.10.2014]. Available: http://www.snowflakesoftware.com/products/gopublisher/workflow/.

[37] Dspace – an open source repository application. [WWW]. [Referred 3.10.2014]. Available: http://dspace.org/.

[38] SourceForge: DSpace source code. [WWW]. [Referred 3.10.2014]. Available: http://sourceforge.net/projects/dspace/files/DSpace%20Stable/4.2/.

[39] GitHub: DSpace source code. [WWW]. [Referred 3.10.2014]. Available: https://github.com/DSpace/DSpace.

[40] Duraspace: DSpace documentation. [WWW]. [Referred 3.10.2014]. Available: https://wiki.duraspace.org/display/DSDOC4x/DSpace+4.x+Documentation.

[41] GNU Affero General Public License. [WWW]. [Referred 8.10.2014]. Available: http://www.gnu.org/licenses/agpl-3.0.html.

[42] Delanghe, S. Using Learning Styles in Software Documentation. IEEE Transactions on Professional Communication, Vol. 43, no. 2, June 2000, pp. 201-205.

[43] Ganier, F. Factors Affecting the Processing of Procedural Instructions: Implications for Document Design. IEEE Transactions on Professional Communication, Vol. 47, no. 1, March 2004, pp. 15-26.

[44] Webb, Donna, R., (1989). Writing Effective User Manuals: Basic Guidelines and Tips. Library Hi Tech. Vol. 7. no. 4. pp. 41-47.

[45] Sommerville, I. Software Documentation. [WWW]. [Referred 13.10.2014]. Available: http://www.literateprogramming.com/documentation.pdf.

[46] Kennedy, Patrick, M. Technical Writing Tips. Tech Directions. November, 2004, pp. 22-23.

[47] Gitorius. [WWW]. [Referred 23.10.2014]. Available: https://gitorious.org/.

[48] Git. A free and open-source distributed version control system. [WWW]. [Referred 23.10.2014]. Available: http://git-scm.com/.

[49] Mercurial source control tool. [WWW]. [Referred 23.10.2014]. Available: http://mercurial.selenic.com/.

[50] Subversion. [WWW]. [Referred 23.10.2014]. Available: http://subversion.apache.org/.

[51] Bazaar version control system. [WWW]. [Referred 23.10.2014]. Available: http://bazaar.canonical.com/en/.

[52] Apache Allura. Forge software for hosting software projects. [WWW]. [Referred 23.10.2014]. Available: https://allura.apache.org/.

[53] Fogel, K. Producing Open Source Software. How to Run a Successful Free Software Project. [WWW]. [Referred 23.10.2014]. Available: http://producingoss.com/en/index.html.

[54] GNU GPL, version 3. [WWW]. [Referred 23.10.2014]. Available: http://opensource.org/licenses/GPL-3.0.

[55] Apache License, Version 2.0. [WWW]. [Referred 23.10.2014]. Available: http://opensource.org/licenses/Apache-2.0.

[56] Green, S. Somers, F. Software Agents: A Review. [WWW]. [Referred 5.11.2014]. Available: https://www.scss.tcd.ie/publications/tech-reports/reports.97/TCD-CS-1997-06.pdf. pp. 26-39.

[57] Rosen, L. Open Source Licensing: Software Freedom and Intellectual Property Law. Chapter 10: Choosing an Open Source License. [WWW]. [Referred 11.11.2014]. Available: http://flylib.com/books/en/4.467.1.1/1/.

[58] Academic Free License. [WWW]. [Referred 12.11.2014]. Available: http://opensource.org/licenses/AFL-3.0.

[59] Open Software License. [WWW]. [Referred 12.11.2014]. Available: http://opensource.org/licenses/OSL-3.0.

[60] Common Public License. [WWW]. [Referred 12.11.2014]. Available: http://opensource.org/licenses/cpl1.0.php.

[61] Hecker, F. Setting Up Shop: The Business of Open Source Software. IEEE Software, January, 1999, pp. 45-51.

[62] Maxwell, E. Open Standards, Open Source, and Open Innovation: Harnessing the Benefits of Openness. Mit Press Journals, Innovations, 2006, pp. 138-141.

[63] Bilen, C., C. Alavizadeh, Z. Open Source Strategy: A Change of Perception trough the Lens of Innovation. Master Thesis within Business Administration. March, 2011. pp. 29-30, 41-51.

[64] The BSD 3-Clause License. [WWW]. [Referred 14.11.2014]. Available: http://opensource.org/licenses/BSD-3-Clause.

[65] The GPL License. [WWW]. [Referred 14.11.2014]. Available: http://opensource.org/licenses/GPL-2.0.

[66] Krishnamurthy, S. An Analysis of Open Source Business Models. February 2003. [WWW]. [Referred 14.11.2014]. Available: http://faculty.washington.edu/sandeep/d/bazaar.pdf.

[67] Codehaus Licenses Page. [WWW]. [Referred 17.11.2014]. Available: http://www.codehaus.org/customs/licenses.html.

[68] AVAA open-access publishing platform. [WWW]. [Referred 18.11.2014]. Available: https://www.tdata.fi/avaa.

[69] AVAA Pilot portal. [WWW]. [Referred 18.11.2014]. Available: http://avaa.tdata.fi/web/avaa/etusivu?p_p_id=56_INSTANCE_WnrBJ3tLtuRh&p_p_lifecycle=0&p_p_state=normal&p_p_mode=view&p_p_col_id=column-

1&p_p_col_count=1&_56_INSTANCE_WnrBJ3tLtuRh_languageId=en_US.

[70] IDA Storage Service. [WWW]. [Referred 18.11.2014]. Available: https://www.tdata.fi/ida.

[71] AVAA Data&applications – Geosciences. [WWW]. [Referred 18.11.2014]. Available: http://avaa.tdata.fi/web/avaa/geotieteet.

[72] WebDAV. [WWW]. [Referred 18.11.2014]. Available: http://www.webdav.org/.

[73] CSC – IT Center for Science. [WWW]. [Referred 19.11.2014]. Available: https://www.csc.fi/home.

[74] iRODS – The Integrated Rule Oriented Data System. [WWW]. [Referred 19.11.2014]. Available: https://irods.org/.

[75] CoRed – Collaborative Coding in the Web for the Web. [WWW]. [Referred 26.11.2014]. Available: http://cored.cs.tut.fi/.

[76] GitHub – CoRed Source Code. [WWW]. [Referred 26.11.2014]. Available: https://github.com/ahn/cored.

[77] GitHub Press Page. [WWW]. [Referred 4.2.2014]. Available: https://github.com/about/press.