HTML5-based Mobile Agents for Web-of-Things

Jari-Pekka Voutilainen, Anna-Liisa Mattila, Kari Systä and Tommi Mikkonen
Tampere University of Technology, Korkeakoulunkatu 1, FI-33720 Tampere, Finland
E-mail: first.last@tut.fi

Keywords: web applications, mobile agents, Internet-of-Things, Web-of-Things, HTML5, JavaScript

Received: July 24, 2015

Systems and services utilizing Internet-of-Things can benefit from dynamically updated software in a significant way. In this paper we show how the most advanced variant of moving code, mobile agents, can be used for operating and managing Internet-connected systems composed of gadgets, sensors and actuators. We believe that the use of mobile agents brings several benefits, for example, mobile agents help to reduce the network load, overcome network latency, and encapsulate protocols. In addition, they can perform autonomous tasks that would otherwise require extensive configuration. The need for moving agents is even more significant if the applications and other factors of the overall experience should follow the user to new contexts. When multiple agents are used to provide the user with services, some mechanisms to manage the agents are needed. In the context of Internet-of-Things such management should reflect the physical spaces and other relevant contexts. In this paper we describe the technical solutions used in the implementation of the mobile agents, describe two proof concepts and we also compare our solution to related work. We also describe our visions of the future work.

Povzetek: Razvit je sistem mobilnih agentov v HTML5 za splet stvari.

1 Introduction

One of our drivers, the Internet of Things (IoT) refers to an approach where extensive amount of physical objects are inter-connected and also connected to the Internet. When implemented, IoT systems open possibilities for new applications and services for the users. At the moment much of the research has been invested in low-level issues related to addressing the different kinds of devices, bandwidth used in the communication, and latency in communications. However, since the main goal is to enable new applications and services higher-level protocols are also needed. Due to the diverse needs of different applications and services and vast number of different devices the protocols face extensive needs of adaptability. Design of such protocols upfront would assume extensive configurability and in extreme cases extra proxies and other workarounds. If all connected devices can dynamically accept new executable code, the risks are significantly reduced.

The other driver, from the human user point of view is the fact that people use an increasing number of Internet-connected devices to access services and applications from the Internet. This leads to a need to different multi-device experiences and eventually to concept of Liquid Software [23], where the user can effortlessly use multiple devices to access their applications and content from different devices in different contexts. Liquid Software, as described in [23] concentrates in systems where end-user devices with screens interact with Internet services. In this paper we show how the ideas of Liquid Software and Mobile Agents, as one building block to implement Liquid Software, can be applied in the world of Internet of Things.

Many researchers, for instance [10] separate Web of Things (WoT) from Internet of Things (IoT), because the former is based on resource-based APIs, resource-oriented architecture (ROA) and RESTful paradigm [6], and the latter is based on approaches that reflect the remote procedure call (RPC) paradigm. The main purpose of both approaches is the same: to connect devices around us to Internet and to use them in providing value to users. The difference is in the architectural approach, and because we share the architectural approach of WoT we use term WoT (Web of Things) in this paper.

In this paper we present our framework where HTML5 based mobile agents are used for programming WoT. The framework contains an agent framework that enables the usual operations associated with mobile applications, an application model for creating such agents, and a management system that is based on physical spaces and other real-world concepts.

This paper summarizes our earlier work on mobile agents [11], [12], [14] and [22], but also reports new work, for example, new way of separating user interface from the agent logic and for management of the agent system – including mobility of the agents.

The rest of this paper is structured as follows. After background and motivation in Section 2, we introduce our mobile agent framework and its implementation, and programming framework in Section 3. This description is based on older publication [22], but significantly reorganized and updated to reflect our latest design including new features like Management server and new declarative way to handle UI. Especially in Subsections 3.5 and 3.6 we discuss how a “thing” can host agents,

what operations the agent can perform and how the system can be organized in Cloud Spaces. In Section 4 we present some experiments we have done with the system. In Section 5, we briefly address related work. In Section 6 we discuss current state of our work and our vision of future work. Finally in Section 7 we draw some final conclusions.

2 Motivation and background

Mobile Agents are executable entities that can move from one node to another together with the internal state of the application. This means that an executing agent can pause its execution in current location and then continue in a new location. In fact, mobile agents represent a special case of moving code combining remote evaluation with preservation of the internal state. The mobile agents discussed in this paper can preserve the internal state if the application needs that functionality. In some cases, we just need to send the code for remote evaluation.

Mobile agents have certain benefits that we see especially useful for Internet of Things. Among the benefits listed in [13], the following have special relevance in the scope of IoT:

- *Mobile agents reduce the network load.* Many "things" include sensors that monitor physical environments and thus potentially generate hidden data flows. If all that data is sent to application on another end of the network, the network may be flooded with data. A mobile agent running in the thing can reduce the network load by pre-processing the data generated by the sensors.
- *Mobile agents overcome network latency. The* latencies of networks, especially in wireless networks, can make real-time control impossible. Thus, everything cannot be done in the cloud and local execution is needed.
- *Mobile agents encapsulate protocols.* New protocols get invented frequently and objects in IoT should adapt to those. Agents are good tools for introducing new protocols or data formats.
- *Mobile agents execute asynchronously and autonomously.* This means that there is no need to generate network traffic for every execution. In case of wireless networks this also reduces power consumption.

As already pointed out, we propose using mobile agents in the context of WoT [12]. Our mobile agents are based on web technologies. An agent can move between different devices, and if necessary it is also possible to clone agents to create more instances. This enables the creation of increasingly complex configurations, where device- and context-specific decisions can also be taken in devices.

The Liquid Software dimension of our research is related to dynamically moving applications that enable use of several devices for accessing and controlling the WoT systems. The idea is that the execution should dynamically move to a location where it can be done more efficiently and where the required resources are. On the other hand, things that matter to the user, like user interfaces and user content should follow the user whenever possible and be accessible with device that user happens to have in her hand at that moment.

Third aspect of this paper is organization of the agent platform to "spaces" that relate to physical spaces and other real-world contexts. These Cloud Spaces define management structure for the WoT systems.

3 Architecture and concepts

Our whole system is based on mobile agents implemented with HTML5 technologies. This framework has originally been described in [22], but the design has evolved since then. Subsections 3.1, 3.2 and 3.3 report our current design, including new technique to separate UI from the logic, and framework for external control of the agents. The mobile agent framework is then in subsection 3.5 applied to Internet of Things by bringing agent servers close to various devices [12]. Finally, in subsection 3.6 we show how the systems are organized to managing contexts called Cloud Spaces [14].

3.1 Execution of HTML5 agents

In our design, an HTML5[1] agent is an HTML5 application that can run in two modes, with a user interface inside a browser and in a headless mode, that is, without a user interface, in an environment called Agent Server [22]. For executing the agent headlessly in the Agent Server, only a JavaScript virtual machine with a simple runtime environment is required. No full browser is needed. The state of the agent is saved during the migration between server and browser and the agent continues its execution as if there wasn't any change in the mode.

During its life cycle the agent may visit several browsers and several Agent Servers. An example life cycle is presented in Figure 1. The instance of an agent is created when it is downloaded from the Origin Server. This server is similar to an ordinary web server, and its task is simply to host applications. After the download, the executing agent can move to an Agent Server to continue its execution and back to a browser again.

The Origin Server maintains and serves all the files, and when the agent moves between Agent Servers and Browsers we usually deliver only the URL that point to a resource in the Origin Server. The receiving entity then fetches the static content from Origin Server.

The dashed box "Mgmt. server" and the dashed arrows in Figure 1 depict an optional management functionality that allows external entities to control agents.

Our all protocols are Web-friendly and rely on standard HTTP. Both Origin Server and Agent Server are

[1] For the purposes of this paper, the overall goal of HTML5 to support rich applications is important; we do not refer to any specific new technology introduced by HTML5.

HTTP servers that can be accessed with HTTP requests. Agents are fetched for execution with GET and pushed to server with POST. This means that an agent can also move from one server to another. In addition, the Agent Server can provide a list of running agents. Concretely speaking, the most important parts of the HTTP interface of the Agent Server are:

- */list* (HTTP GET) gets a list of active agents as an HTML file that can be shown in a browser.
- */upload* (HTTP POST) sends URLs to agent code and user interface together with serialized state. After receiving the Agent Server instantiates and starts the agent.
- */<id>* (HTTP GET) pauses the agent in server, serializes the state and sends it to the requesting browser

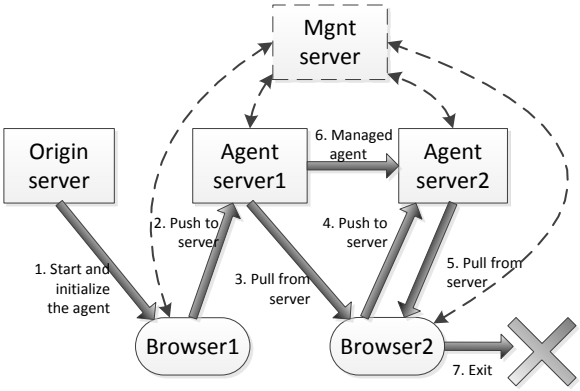

Figure 1. Life cycle of an HTML5 agent in the framework.

As usually in today's web applications, the HTML file of the agent includes references to Cascading Style Sheets (CSS), to other HTML files, images and other resources, and JavaScript files. Also, similarly to standard web applications the agent is first started by downloading the HTML file from the origin server.

Agents are serialized whenever they are moved between servers or between a server and a browser. The implementation of the framework provides mechanism for serialization of the relevant parts of the state. When an agent is about to move to a new location its state is serialized into JavaScript Object Notation (JSON) based on state variables defined by developer. An example of serialized agent description is shown below, where state of this agent includes four variables *low*, *high*, *count* and *history*:

```
{"auri":
  "http://xx.xx.xx.fi:pppp/gmonitor.js",
 "huri":
  http://xx.xx.xx.fi:pppp/gmonitor.html
 "id"    : "526636" ,
 "memory": {
   "high"   : 0.0253 ,
   "low"    : 0.0214 ,
   "count"  : 3,
   "history": [0.0253, 0.0234 ,0.0214]}}
```

This serialization contains URIs for the agent functionality (JavaScript file) and HTML based UI. In addition it has a unique identity variable (id) and set relevant variables in application state encoded in JSON dictionary "memory".

When an Agent server receives the serialized agent description, it fetches the JavaScript code from the address in *auri*, in addition, the Agent downloads the other JavaScript files implementing the framework, it initializes the agent using the serialized state, and finally it starts the execution of the Agent.

When a browser requests the agent from the Agent Server some special arrangements are needed due to security and other limitations of the browsers. As response to a request from browser to the Agent Server, the Agent Server sends the content of HTML-file identified by 'huri' field of agent description. To that HTML file the Agent Server injects JavaScript to restore the transferred local state of the agent,

The execution model of the agent also needs to be suitable for the execution environment. First of all it needs to be suitable for running in the browser. For instance it should not block the event loop of the browser run-time. On the other hand it needs to proceed without user interface events delivered by the browser. Furthermore, the agent needs to have safe points in execution so that a consistent state can be serialized. In practice this means that all the application logic is embedded in specific event handlers that are triggered by timer events. This event-based execution model fits well to Agent Servers that have been implemented with Node.js [19].

3.2 Management API

The management protocol is also made compatible with the overall design. The Management Server implements a REST interface for both the moving agents and a control application. The control application may be operated by a human user, or be an autonomously running application. The most important part of the API for agents consists of two kinds of REST calls: "ImHere" when the agent has arrived to a new location, and "Status" call is sent to the Management Server on regular intervals. The response to these REST calls may contain an instruction to the agent to move to a new location (see arrow 6 in Figure 1). Our current implementation includes also instructions for the application to exit and to change values of variables. Control applications can browse the agents and their histories. Control applications can also send instructions to the Agent.

In the following we give a short example. When control application makes a GET request to */Agents* it gets a list of agent IDs as a response.

```
GET http://host/Agents => [211, 311]
```

In this case the Management Server knows about two agents. Detailed information about a specific agent can be retrieved with

```
GET http://host/Agents/211 => {…}
```

The response includes information about the location and status of the agent. The control application can request an agent to move to a new location by sending payload `[{"goto": http://server2}]` by using a PUT request

```
PUT http://host/Agents/211/todo
```

This request is now waiting in the Management Server until Agent 211 contacts the Management Server. When the agent 211 updates its status by sending

```
{"id":"211", "Status":"I'm fine"}
```

with request

```
PUT http://host/Management/Status
```

to the Management Server, the request to move to new location is delivered to agent 211 in the response, and the Agent framework initiates the move to the requested location.

This is a lightweight management framework that assumes the agents co-operate and does not affect agents that do not participate. The REST API of the Management Server has been designed both for automated control and for management user interface described in Subsection 4.2. On the other hand the framework relies on basic HTTP protocol and thus does not require the infrastructure to support any other protocol. In the current implementation only the agents that are in server obey all received instructions and agents that are in browser ignore the requests to move.

3.3 Programming agents

Core parts of the Agent framework have been encapsulated in a reusable JavaScript class Agent, and the developer should specialize her own version from that class. So far we have used the functional inheritance pattern presented in [3], but the more traditional prototype inheritance could be used, too. The application-specific sub-class of the Agent can override the following methods:

- Method *getRunningStatus()* – should return a string that the management interface of the agent server context can show.
- Method *preupload()* is called by upload() just before serialization as the first the uploading. By overriding this method the agent can implement application specific preparations for the uploading.
- Function *continueWork()* – re-initializes the execution when the agent has arrived and de-serialized in a new location and the execution should be resumed. This function initializes the state of the agent by recreating the variables.

In addition, the agent has to provide a function that creates and initializes the agent object.

The framework provides also a set of utility methods that the above methods and functions can call. The most important utility methods are:

- *registerVar(name*) – with this function the application can state that a variable is part of relevant local state and will be automatically serialized.
- *setWork(function, interval)* – sets the work function that is periodically executed with the given interval. The framework assumes that the work function returns reasonable quickly.
- upload(url) – uploads the agent to an Agent Server specified by parameter url. This function first stops execution, then serializes the state and finally sends the serialized agent to the Agent Server.

As discussed earlier, the agents run in the headless mode in Agent Server and with the HTML and CSS files in browser. This means that the JavaScript code of the agents has to be written to be executable without presence of the complete Document Object Model (DOM) tree. Separation of the application logic from the UI part is not always easy since many Web application frameworks rely on existence of the DOM-tree. In our first implementation we assumed that agents are written for the framework so that user interface is nicely separated from the application logic. In addition, we provided a very simple DOM emulation to help writing of portable applications. We have later experimented with a different approach to help application developers in implementation of Agents that can run with and without DOM. This approach is based of declaration of the binding between application logic and user interface with primitives like:

```
BindModeltoView(['var','elem']);
BindModeltoView(['var.func', 'elem']);
```

The first declaration states that if element with id 'elem' exists in DOM-tree its value (innerHTML) is updated with the value of variable 'var' whenever value of var changes. In the latter version function 'func' is used instead of simple assignment to innerHTML of 'elem'. If the above binding mechanism is used, the application code does not need include UI-specific code and thus there is no need to deal with differences between server and client since the framework includes conditional code.

3.4 Agent communication

A simple agent-to-agent communication framework has also been implemented [11]. This framework allows agent to send messages and to receive their messages regardless of their current location. Because the web infrastructure does not support communication between browsers, the all communication is routed through an agent server.

With the current API, the sending agent initializes the communication as follows:

```
c = new CommComponent(function(msg) {
  …
});
c.setNameSpace("myChannel");
c.initIO();
```

and a message is sent with:

```
c.sendMessage(obj)
```

Like in other parts of our framework the content is sent over the network as a JSON string. The namespace "myChannel" is kind of channel and the receiving end can listen the channel with the following code:

```
c = new CommComponent(function(msg){
    // process incoming msg
});
c.setNameSpace("myChannel");
c.initIO();
```

We have not used this framework much in our applications still, because most of our example applications have assumed that the moving agents bring the data with them. We have just validated that our implementation that is based on WebSockets [25] works.

3.5 Agent servers in "things"

As described earlier, the core components of our Agent Server are the HTTP server and a virtual machine executing JavaScript. These can be implemented, for example, with Node.js [19] technology. The agent server has two main functions: 1) implement execution environment that is compatible enough with the browser and 2) simple management function for agents.

As the implementation of the Agent Server only requires Node.js and a few hundred lines of JavaScript code and because our Agent Server does not require lot of computational resources, and it can be included in many small devices – or "things" – that are connected to the Internet. In our experiments we have used a low-cost single board computer Raspberry PI [20], which typically runs Linux. The infrastructure requirements are equal to those of WoT, because the devices are accessed with standard HTTP requests such as GET and POST. With these requirements the Raspberry PI device goes beyond the bar with a clear margin.

We assume that most devices that can be nodes in SOA based IoT or REST-based WoT can also host our Agent Server. This would bring benefit of mobile agents described above, but also enable new ways for remote management and extending the functionality by adding new code in a form of mobile agents. The possible application areas include the following:

- *Home automation that goes beyond remote control*. An intelligent agent can work on behalf of the user and implement even complex strategies to optimize energy consumption and user comfort.
- *Support for new communication protocols or applications*. In most cases the applications are in the Internet, but the "things" need to be accessible. Sometimes new application will need new functionality from the devices.
- *Compatible extensions to already existing systems*. Interoperability with new devices may be achieved by adding new intelligence to existing devices.

The proposed approach has obvious benefits over the solutions that have been more conventionally used. From the perspective of the "thing" executing the agent, the agent framework based on managed runtime effectively creates a sandbox that separates the agent from the rest of the system. Therefore it is, for instance, possible to run real-time critical code in the same system, and only execute the agent when there is leftover execution time, a partitioning which is supported by many real-time operating systems. A further benefit over other agent frameworks is that we are solely relying on web protocols and technologies. The ecosystem that builds web applications has presently advanced to a level where the web is increasingly a platform for all applications. Allowing this ecosystem to build mobile agents for WoT creates low-hanging opportunities, because there is no need to invest in familiarizing yet another platform.

3.6 Structuring of the framework

In [14], we have presented and demonstrated a structuring concept that incorporates HTML5 agents and their servers with a data solution to store user's content. Each Cloud Space is essentially a private cloud which hosts user's data and applications. In the context of WoT, Cloud Spaces can be seen as ecosystem where each "thing" provides small functionality for the Cloud Space as a whole. Data streams between the nodes can be implemented using agents and when adding a new node to the system, agents can provide architecture configuration automatically to the new node. Figure 2 depicts on possible Cloud Space configuration with Web of Things.

Each "thing" in WoT implements minimum of the Agent Server architecture and when new device is added to WoT, agent is sent to the new device to configure it. For example agent creates new public interfaces to the new device, which can then be used for data streams. Or the agent can implement application logic for the device, which is then executed even without the agent. Even if the agent does not modify the programming of the new device, it can provide information about WoT, for example, location of other servers and devices.

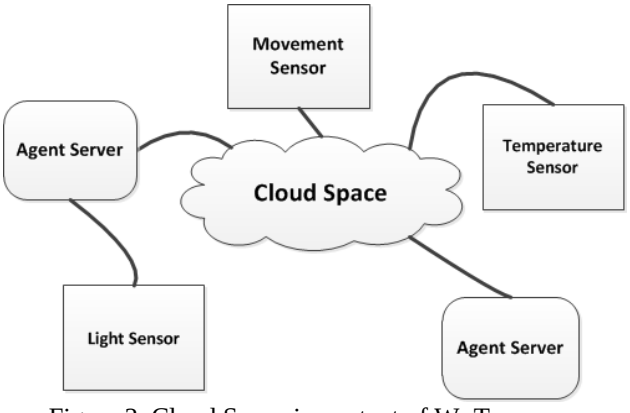

Figure 2. Cloud Space in context of WoT.

4 Proof-of-concept experiments

During our research we have implemented a few proofs of concept and demonstrators. Here we describe two demonstrators that relate to devices in WoT and to management of the agents and Cloud Spaces.

4.1 Agents for embedded device

To verify and demonstrate our idea we implemented a simple agent that collected information from different sensors hosted by different devices (Figure 3).

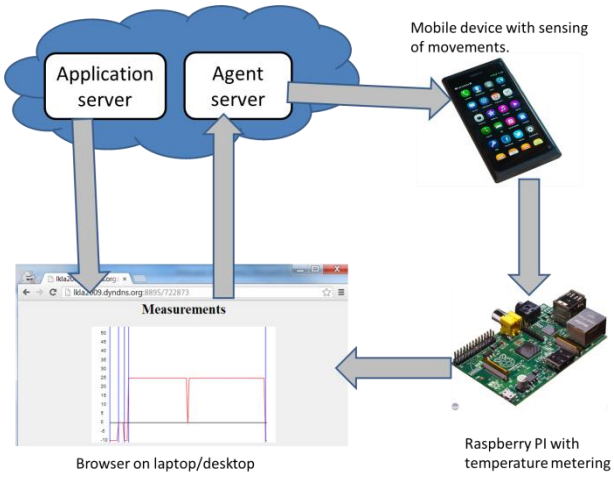

Figure 3. A traveling agent in different devices.

The implementation is based on the following components.

1. Two Agent Servers, one running in Raspberry PI, and another in a standard Linux server running on a virtual machine in a cloud. Both servers are based on Node.js [19] technology, and the implementation of the agent framework is the same in both servers. It is also possible to connect external sensors and actuators to Raspberry PI. In our case, we have connected a temperature sensor DS18B20 [15] for our experiments.
2. An agent that travels between servers and browsing devices. The agent is written in a manner that it can measure temperature when it is in the Raspberry-hosted server and if temperature sensor is available. When in browser the agent receives DOM device orientation events [24] and measures the orientation of the device. Based on the orientation events, the agent also calculates a "restless index" – i.e. how much the device is rocked or shaken lately. In other words, the agent collects different data in different devices but remembers and aggregates all the collected data to a pre-processed form.
3. Visualization of the collected data when the agent is in browser. In this visualization we show graphs of the temperature and restless index over time.

In the scenario depicted in Figure 3 the agent is first downloaded to a browser running on a Windows laptop. From there it is pushed to an Agent Server in the cloud. The graphical display disappears but collected statistics are preserved and the agent continues its execution. Unfortunately no sensors were available, so no real data was collected. Next the agent is downloaded to a browser running on a smart phone, there the graphical visualization is generated again and the user can see from the graph what has been measured and collected. From mobile browser the agent is uploaded into an Agent Server in Raspberry PI. From there the agent is finally downloaded to a desktop browser.

The purpose of this experiment was to validate that the agent runs in all needed hosts and also to demonstrate the idea. An example – a different execution from the one shown in Figure 3 – of the visualization has been given in Figure 4. The X-axis in Figure 4 represent time (concrete values not shown in Figure 4) and Y-axis show the sensor values.

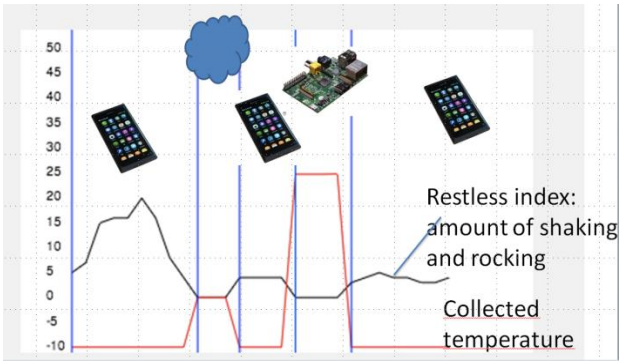

Figure 4. Visualization of the collected sensor data.

Additional text and images have been added to the picture to improve the presentation in this paper. The blue vertical lines indicate moves from one location to another. The history of events in this example run is the following: the agent was first downloaded to browser in a smart phone. Since accelerator sensors were available the restless index gets calculated and recorded, but because temperature sensor is not accessible, temperature defaults to -10 degrees C. The agent is next pushed to an Agent Server in the cloud where neither sensor is available and both readings default to 0. Then the agent is downloaded back to a mobile browser and further pushed to a server in Raspberry PI. In Raspberry PI the agent reads and collects temperature data until it gets downloaded back to mobile browser.

4.2 Managing agents in cloud space

For combining concept of agents and Cloud Space we have also implemented a proof of concept manager for agents running in Cloud Space [14]. The manager is a web application with a 3D interface for managing agents in Cloud Space contexts. By using the manager the user can access to her Cloud Space context and agent servers inside the Cloud Space. User can fetch agents from servers to her web browser and move agents from a server to another server even between contexts.

As Cloud Space context can represent a physical place, panoramic photo spheres are used to visualize the context in the 3D management UI. A real-world image helps the user in mapping of the concepts of Cloud Space to the physical space. Agent Servers in a context are represented as 3D grids and agents running in a server are shown as cuboids are placed to the grid. User performs all management actions, e.g. navigating in contexts, moving agents and fetching agents, via direct manipulation using mouse and keyboard.

Figure 5 presents the management UI in action. In the top section of Figure 5 (marked with 1) the user has dragged the agent on top of the context which she wants to move the agent. When she releases the agent the Management View changes to the context she chose. This is visualized in the middle section of Figure 5. Finally the user can drag the agent to the server in the context and the agent is moved there (bottom section in Figure 5).

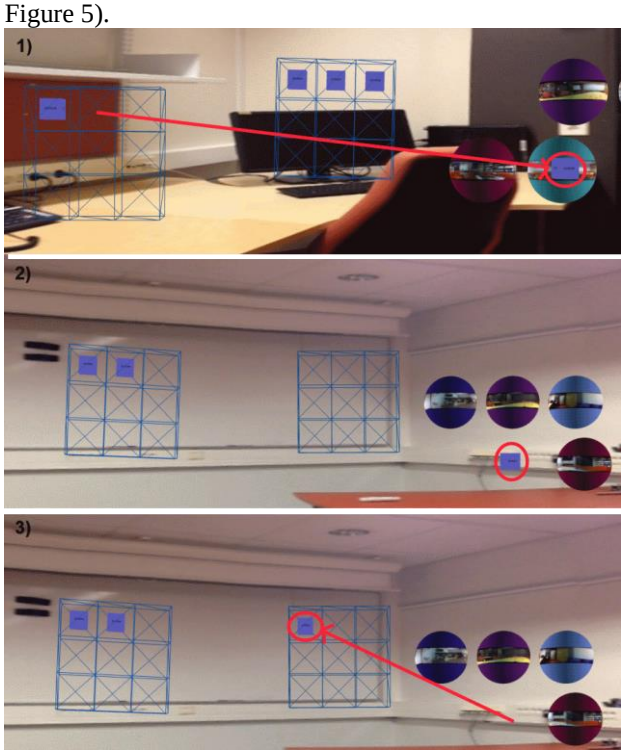

Figure 5. Examples of management views [14].

5 Related work

Use of Web and HTML as an agent platform is not very common. The Radigost system [16] [17] uses Web and JavaScript as an implementation platform for multi-agent systems. It has many interesting features like support for standard agent communication mechanisms and yellow-pages service for agent directories. However, it does not support dynamically moving agents or running agents outside browser. Radigost has later been merged with JavaEE-based agent framework that allows execution of agents also in the server side. In contrast to many benefits of JavaEE-based agent platform, it cannot be hosted on small devices as required by WoT applications. Another Web-based agent-framework has been described in [5]. In that concept the agent platform is based on concepts of Pneuna that is relatively close to our agent description and Soma that is the execution environment. In this approach Soma hides the differences of browser and server environment and creates a completely new application platform for mobile agents. In our approach standard and well-known HTML5 is the agent platform. In addition, the approach presented in [5] has not been designed for pushing agents to agent server when user or browser is not active or when the agent should find a new browser to run on.

As discussed earlier, Web of Things (WoT) and Internet of Things (IoT) approaches lead to a bit different architectures. Because the former leads to resource-oriented architecture (ROA) and the latter is based on approaches that reflect the remote procedure call (RPC) paradigm. While our work could be connected to both approaches, we propose a third approach that is based in sending code for execution in or close to a "thing". In the categorization of moving code proposed in [1], this is called Remote Evaluation. The code sent to remote host can expose new interfaces either in WoT or IoT style. As our system allows executing code to move with its internal state and because the code and state can further move to a yet another location, our system fulfills the criteria of mobile agents. For many WoT and IoT applications, the core subset of mobile agent behavior – remote evaluation – is enough, but moving with state and ability to move even further are available for those applications that benefit from it.

There are a few approaches that support uploading and remote evaluation of code in a "thing". For example, MoteLab [27] is a test bed for sensor networks. The developers using MoteLab can upload executable Java code with a job description towards a "thing". The Web interface is a separate system based on PHP. Somewhat similar system is Kansei [4] – later refactored to KanseiGenie – where developers can also create jobs to execute sensor applications. Our system can also be used in a similar way and from similar motivations. However, in our system the uploaded code is Web content and we can upload an executing agent with its internal state.

Use of web technologies to for IoT or WoT applications is not new. For example, WebIoT [1] is based on similarities to our work. Similarities lie especially in the aim to bring IoT to Web 2.0 and allowing users to develop, deploy and execute their own applications. However, WebIoT does not support agent model.

Maybe the most similar approach to us is the mobile agent framework proposed in [6]. It provides nodes in heterogeneous device networks with a way to communicate and co-operate. Furthermore, it provides means to proactively search for required resources. The system is based on Java-based AgentSpace [21] mobile agent platform. At the moment we do not have similar automatic searching – this is left for future work. On the other hand we have the unique benefits of using the Web, which enables leveraging the power of the web development ecosystem in application development [22].

6 Discussion and future work

The ability to send code for remote evaluation and especially mobile agents is useful when implementing new types of IoT applications. This approach increases flexibility of the system design and evolution of IoT since the new code can add new functionality and adapt the device to new requirements. Moving code and especially agents can also be used to add autonomous intelligence to systems.

Our example agent collected data that was available at the particular location of execution and different data

was collected in different locations. So, the agent adapts to its execution environment. One benefit of mobile agent is reduction of communication. In our case the sensor data, like temperature measurement, was continuously collected but sent over network only when agent moves. Furthermore, calculation of the restless index is an example of agent that reduces communicated data by pre-processing the raw data.

We see that use of web technologies as a basis for our agent framework gives us several benefits. First of all we gain ecosystem benefits in terms of competencies, training material and tools. Secondly, any device with a reasonable recent browser can be used to run and control the agents. Thirdly, web-based agents can be run both in "things" and servers in the cloud, and integrate well with the infrastructure of the Internet.

In the future we would like to study the opportunities when combining our mobile agents to RESTful or SOA paradigms more closely. For instance, and an agent that is located in a "thing" with a temperature sensor can expose a REST API for applications to ask current temperature, list of recent measurements, or some other information, depending on the application needs.

Mapping our framework to agent-related standards is also a potential future topic. Since our agent-to-agent communication solution is very generic, we assume that it can be used as a transport layer to FIPA Agent Communication language [7] in a similar manner as in Radigost [17], but the mapping between our management system and corresponding FIPA standard [8] requires some analysis.

So far we have tried the framework only in reasonable small cases, and one of the most important topics for future work is testing it in a larger context. One potential case is experience roaming with Liquid Software [23]. Mobile agents would allow users to bring their preferences and on-going work to the physical smart spaces they enter. For instance the user can bring his lightning, heating and other preferences to hotel rooms while they travel and they can also use their favourite user interface in favourite mobile device to monitor and control devices in the visited environment.

Another possible experimentation would involve a system that moves agents autonomously. Especially in sensor network systems, automatic crawling of agents could allow them autonomously search and collect the needed data. The recently added management API (see Subsection 3.2) and its underlying mechanisms provide a basis for this such experiment. One of the design goals of the management API was to support such autonomy. Since this API is still a reasonable new feature our framework and we need experiment with it by implementing some example application. Moreover, some obvious things that require attention are related to non-functional properties of our agent system, including scalability and security in particular.

7 Conclusions

In summary, we believe that by the end of this decade multi-device usage will become so seamless and ubiquitous that "it will weave itself into the fabric of everyday life until it is indistinguishable from it" [26]. In contrast to numerous platform and vendor-specific systems, our work on HTML5 agents and related infrastructure demonstrates that such future can be created with technologies that reflect Open Web principles laid out in the *Mozilla Manifesto* [18]. Built with technologies that are open, accessible and as interoperable as possible, and run in standards compatible web browser without plugins, extensions or additional runtimes, they require no installation or manual upgrades, and they can be deployed instantly worldwide, and allow application development and instant worldwide deployment without middlemen, distributors, or platform-specific app stores.

We believe that these properties will be key characteristics for the IoT and WoT devices of the future as well. When these properties are extended to devices, the devices can be part of the new and unified computing infrastructure defined by the Internet.

In particular, we believe that mobile agents can play a special role of connecting devices to the Internet and in allowing the most efficient use of them in the world where everything becomes Internet-connected.

8 References

[1] 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. Pages 22-32.

[2] Castellani, A.P., Dissegna, M., Bui, N., Zorzi, M., WebIoT: A Web Application Framework for the Internet of Things, Wireless Communications and Networking Conference Workshops (WCNCW), 2012 IEEE, Paris France, 2012, pages 202 – 207.

[3] Crockford, D.: JavaScript: the Good Parts, O'Reilly Media, Inc. May 8, 2008.

[4] Ertin E., Arora A., Ramnath R., Naik V., Bapat S., Kulathumani V., Sridharan M., Zhang H., Cao H., and Nesterenko M., "Kansei: a testbedfor sensing at scale," in ACM/IEEE IPSN, Nashville, Tennessee, USA,2006, pp. Pages 399–406.

[5] Feldman, M.: An approach for using the Web as a Mobile Agent infrastructure, In pro-ceedings of the International Multiconference on Computer Science and Information Technology, pp. 39 – 45, 2007.

[6] Fielding R.. Architectural styles and the design of network-based software architectures. Doctoral Dissertation. University of California, 2000.

[7] FIPA Agent Communication Language Specifications, http://www.fipa.org/repository/aclspecs.html, last visited 2.2.2015.

[8] FIPA, Agent Management Specifications, http://www.fipa.org/repository/managementspecs..html, last visited 2.2.2015.

[9] Godfrey W. W., Jha S. S., B. Nair S. B., On A Mobile Agent Framework for an Internet of Things,

In proceeding of: International Conference on Communication System and Technologies, CSNT 2013, 05-08 April 2013, Gwalior, India, At Gwalior, India. Pages 345 – 350.

[10] Hong Y, A Resource-Oriented Middleware Framework for Heterogeneous Internet of Things, International conference on Cloud and Service Computing (CSC), 2012, Shanghai, China, Pages 12-16.

[11] Järvenpää, L., Development and evaluation of HTML5 agent framework. Master of Science Thesis, Tampere University Technology, 2013.

[12] Järvenpää, L., Lintinen, M., Mattila, A-L., Mikkonen, T., Systä, K, Voutilainen, J-P. Mobile Agents for the Internet of Things, In WASA2013, 3rd Workshop on Applications of Software Agents, Sinaia, Romania, October 11-13, 2013.

[13] Lange, D., B., Oshima, M., 1999. Seven good reasons for mobile agents, In Communications of the ACM, Volume 42 Issue 3, March 1999, Pages 88 – 89.

[14] Mattila, A.L., Systä, K., Mikkonen, T. and Voutilainen, J.-P., Cloud Space – Web-based Smart Space with Management UI, A short paper in 10th International Conference on Web Information Systems and Technologies (WEBIST), Barcelona 3-5, April, 2014.

[15] Maxim Integrated, DS18B20, Programmable Resolution 1-Wire Digital Thermometer, Datasheet. http://datasheets.maximintegrated.com/en/ds/DS18B20.pdf

[16] Mitrović, D., Ivanović, M., Budimac, Z., Vidaković M., Radigost: Interoperable web-based multi-agent platform, The Journal of Systems and Software 90, 2014. Pages 167–178.

[17] Mitrović, D., Ivanović, M., and Bădică, C., Delivering the multiagent technology to end-users through the web. In *Proceedings of the 4th International Conference on Web Intelligence, Mining and Semantics (WIMS14)* (WIMS '14). ACM, New York, NY, USA, 2014.

[18] MOZILLA-MF Mozilla, Inc., The Mozilla Manifesto. URL: http://www.mozilla.org/en-US/about/manifesto/.

[19] Node.js. Web page for document and download of nodejs technology, http://nodejs.org/. Last viewed 03.02.2013.

[20] Raspberry PI web page, http://www.raspberrypi.org, Last visited 28.6.2013

[21] Silva A., Da Silva M., An Overview of AgentSpace: A Next-Generation Mobile Agent System, In Proceedings of the Mobile Agents'98, Springer, 1998. Pages 148 – 159.

[22] Systä, K., Mikkonen, T., Järvenpää, L.,. HTML5 Agents – Mobile Agents for the Web, In the Proceedings of 9th International Conference on Web Information Systems and Technologies (WEBIST), Aachen, Germany, 8-10.5.2013. Pages 37-44.

[23] Taivalsaari, A., Mikkonen, T., Systä, K., Liquid Software Manifesto: The Era of Multiple Device Ownership and Its Implications for Software Architecture. A short paper to appear in 38th Annual IEEE International Computers, Software, and Applications Conference (COMPSAC) in Västerås, Sweden 21–25 July, 2014.

[24] W3C, DeviceOrientation Event Specification, W3C Working Draft 1 December 2011 http://www.w3.org/TR/orientation-event/

[25] W3C, The WebSocket API, W3C Working Draft, 19 April 2011, http://www.w3.org/TR/2011/WD-websockets-20110419/

[26] Weiser, M.,, The Computer for the 21st Century. Scientific American, September 1991, pp. 94-104.

[27] Werner-Allen G., Swieskowski P., and Welsh M., "MoteLab: a wireless sensor network testbed," in ACM/IEEE IPSN, Apr. 2005, Pages 483–488.