An Agent-Based Programming Model For Developing Client-Side Concurrent Web 2.0 Applications

Giulio Piancastelli, Alessandro Ricci, Mattia Minotti

DEIS, Alma Mater Studiorum—Università di Bologna

5th International Conference on Web Information Systems and Technologies (WEBIST 2009)
Lisboa, March 23-26, 2008
Outline

- Background: programming models for complex Web 2.0 applications
- A&A: An Agent Based Programming Model
- An Agent-Oriented Model for Client-Side Web Programming
- simpA-Web Framework Prototype
- A Case Study
  - Definition
  - Design
  - Remarks
- Current Limits and Future Works
Key features of Web 2.0:

- Web as a development platform for *rich applications*
- which allow the client to send multiple concurrent requests in an asynchronous way
- avoiding complete page reload
- keeping the user interface live and responding.
Background - Web 2.0

Key features of Web 2.0:
- Web as a development platform for *rich applications*
- which allow the client to send multiple concurrent requests in an asynchronous way
- avoiding complete page reload
- keeping the user interface live and responding.

Clear advantages in terms of:
- User perceived performance
- Efficiency
- Interactivity.
Current approaches:

- **JavaScript and AJAX:**
  - JavaScript as a single-threaded language
  - Event-driven programming style
  - Request-and-callback mechanisms
Current approaches:

- **JavaScript and AJAX:**
  - JavaScript as a single-threaded language
  - Event-driven programming style
  - Request-and-callback mechanisms

- **Libraries (Concurrent.Thread and WorkerPool API of Google Gears):**
  - thread and process abstractions on top of JavaScript model.
Current approaches:

- **JavaScript and AJAX:**
  - JavaScript as a single-threaded language
  - Event-driven programming style
  - Request-and-callback mechanisms

- **Libraries (Concurrent.Thread and WorkerPool API of Google Gears):**
  - thread and process abstractions on top of JavaScript model.

- **Frameworks (Ruby-on-JavaScript of Rails and Google Web Toolkit):**
  - single object-oriented language for both the client and the server-side of the application.
Problems and Challenges

- **Program complexity**
  - no clear and effective model for programming complex Web 2.0 application featuring concurrency
  - low-level mechanisms for handling concurrency and asynchronous interactions
Problems and Challenges

- **Program complexity**
  - no clear and effective model for programming complex Web 2.0 application featuring concurrency
  - low-level mechanisms for handling concurrency and asynchronous interactions

- **Design complexity**
  - not clear how to design such application
Problems and Challenges

- Program complexity
  - no clear and effective model for programming complex Web 2.0 application featuring concurrency
  - low-level mechanisms for handling concurrency and asynchronous interactions

- Design complexity
  - not clear how to design such application

Contribution of the work

Exploring an uniform programming model with a first-class support for concurrency, asynchronous interaction and distribution.
A&A: An Agent Based Programming Model

- Inspiration from **Activity Theory** and human working environments
  - human actors doing activities in shared context
  - cooperating by talking and sharing and using resources and tools
Applications as **workspaces** composed by **agents** and **artifacts**
- agents → human actors
- artifacts → resources and tools used by humans
- workspaces → shared environments
The basic modeling abstractions are *agents* and *artifacts*. 
The basic modeling abstractions are *agents* and *artifacts*.

- The **agent** abstraction models the (pro-)active part of a system, with autonomous activities targeted to some king of goal, encapsulating then the logic and control of such activities.
  - Actions and perceptions
  - Memory and state
  - Activity-based structure

- The **artifact** abstraction models the resources and tools created and used by agents during their activities.
  - Usage interface
  - Observable events and properties
The basic modeling abstractions are *agents* and *artifacts*.

- The **agent** abstraction models the (pro-)active part of a system, with autonomous activities targeted to some kind of goal, encapsulating then the logic and control of such activities.
  - Actions and perceptions
  - Memory and state
  - Activity-based structure
- The **artifact** abstraction models the resources and tools created and used by agents during their activities.
  - Usage interface
  - Observable events and properties
The basic modeling abstractions are *agents* and *artifacts*.

- The **agent** abstraction models the (pro-)active part of a system, with autonomous activities targeted to some kind of goal, encapsulating then the logic and control of such activities.
  - Actions and perceptions
  - Memory and state
  - Activity-based structure
- The **artifact** abstraction models the resources and tools created and used by agents during their activities.
  - Usage interface
  - Observable events and properties

Besides, the notion of *workspace* provides a logic container of agents and artifacts and can be used to structure the overall sets of entity.
Interaction in terms of Use/Observation

- An agent perform a **use** action over an artifact (a)

(a)

(b)
Interaction in terms of Use/Observation

- An agent performs a use action over an artifact (a).
- The execution of the operation generates observable events and changes the artifact's inner state.
Interaction in terms of Use/Observation

- An agent perform a **use** action over an artifact (a)
- The execution of the operation generates observable events and change the artifact inner state
- Agent sensors collect these events and allow it to retrieve them (b)
Web 2.0 Modelled as Workspaces of Agents and Artifacts

Figura: First Web 2.0 client-side application model.
Methodology

Guidelines for an A&A-based model creation:

1. Identify the basic parts of the system
Methodology

Guidelines for an A&A-based model creation:

1. Identify the basic parts of the system
2. Classify them in
   - task-oriented
   - function-oriented
Methodology

Guidelines for an A&A-based model creation:

1. Identify the basic parts of the system
2. Classify them in
   - task-oriented
   - function-oriented
3. Map each one on A&A meta-model abstractions
   - task-oriented → agents
   - function-oriented → artifacts
Methodology

Guidelines for an A&A-based model creation:

1. Identify the basic parts of the system
2. Classify them in
   - task-oriented
   - function-oriented
3. Map each one on A&A meta-model abstractions
   - task-oriented → agents
   - function-oriented → artifacts
4. Identify the operations structuring artifact functionalities and the activities structuring the agent behaviours.
Model elements:

- **Browser Environment** → **Workspace**
  - Includes every computational activities
  - Agents and artifacts downloaded as a client-side part of web applications will join it
Agent-Oriented Model for Client-Side Web Programming/1

Model elements:

- **Browser Environment → Workspace**
  - Includes every computational activities
  - Agents and artifacts downloaded as a client-side part of web applications will join it

- **Page → Artifact**
  - Accessible, tree-like, standardized DOM object.
  - Allows to dynamically update page content, structure and visualization style.
  - Generates internal events in response to user’s actions.
Model elements:

- **Browser Environment** → Workspace
  - Includes every computational activities
  - Agents and artifacts downloaded as a client-side part of web applications will join it

- **Page** → Artifact
  - Accessible, tree-like, standardized DOM object.
  - Allows to dynamically update page content, structure and visualization style.
  - Generates internal events in response to user’s actions.

- **HTTP Channel** → Artifact
  - Entity for communication through HTTP protocol.
  - Allows to specify the operation to execute, the header values and possibly a payload.
  - Receives responses and makes them available to the other entities in the system.
Common Domain Elements → Agents + Artifacts

- Common components that are frequently found in specific application domains.
  - As an example: e-commerce functionalities.
**Common Domain Elements** → Agents + Artifacts

- Common components that are frequently found in specific application domains.
  - As an example: e-commerce functionalities.

- others...
Figura: Web 2.0 client-side application model.
simpA-Web:  
- extension of *simpA framework*  
- defines agent and artifact templates for client-side Web development
simpA-Web Framework/1

simpA-Web:

- extension of *simpA framework*
  - defines agent and artifact templates for client-side Web development

- *Java Applets*
  - for migration of code
  - signed Applet to drop security constraints (multiple connections towards multiple servers and multithreading)
simpA-Web: 

- extension of \textit{simpA framework}
  - defines agent and artifact templates for client-side Web development
- \textit{Java Applets}
  - for migration of code
  - signed Applet to drop security constraints (multiple connections towards multiple servers and multithreading)
- \textit{LiveConnect}
  - JavaScript library that can be conveniently invoked from the JVM execution context
simpA-Web Framework/1

simpA-Web:

- extension of *simpA framework*
  - defines agent and artifact templates for client-side Web development
- *Java Applets*
  - for migration of code
  - signed Applet to drop security constraints (multiple connections towards multiple servers and multithreading)
- *LiveConnect*
  - JavaScript library that can be conveniently invoked from the JVM execution context
- *Rhinohide*
  - Java library that wrap LiveConnect functionalities for a simpler and more complete support for page events
simpA is an extension of Java to support the main abstractions from the A&A conceptual model
- realized as a library
- exploiting Java 5 annotations

```java
public class MailAgent extends Agent {
    @ACTIVITY_WITH_AGENDA(
        todos={
            @TODO(activity="setup",
                persistent=false),
            @TODO(activity="fetch",
                persistent=false,
                pre=completed(setup)),
            @TODO(activity="check",
                persistent=true)
        }
    ) void main() {}
    @ACTIVITY void setup() { /* ... */ }
    @ACTIVITY void fetch() { /* ... */ }
    @ACTIVITY void check() { /* ... */ }
}
```

```java
public class Page extends Artifact {
    @OBSPROPERTY Document dom;
    @OPERATION void setAttribute(
        String id,
        String attr,
        String val)
    {
        Element e = dom.getElementById(id);
        e.setAttribute(attr, val);
        updateProperty("DOM", dom);
    }
}
```
simpA-Web framework includes two main implemented artifacts:

- **HTTP Channel usage interface**
  - `setDestination`
    - DestinationChanged event
  - `setHeader`
    - HeaderChanged event
  - `send`
    - RequestSent event
    - Response event
    - Failure event

- **Page usage interface**
  - `changeDom`
    - DomChanged event
  - `setElement`
  - `setAttribute`
  - `addChild`
A Sample Agent-Oriented Web 2.0 App - Definition

Scenario:
- N services (type A)
  - offer a list of machine-readable products.
  - allow downloading of an agent which can
    - communicate with the service (using its own protocol)
    - search products with some characteristics.
- A service (type B)
  - offers a static list of type A services
  - allows dynamically searching of type A services.

Client-Side application requirements:
- To search services for a product that satisfies a set of parameters.
- To periodically monitor services.
- To terminate when some user defined conditions are met:
  - quantitative
  - qualitative
  - temporal
Considered application is Web 2.0 relevant because it introduces:

- an open and dynamic scenario
- elements of concurrency
- interaction between heterogeneous entities
- advanced interaction with user interface
- asynchronous communication
- periodic activities

Instead it shouldn’t be considered meaningful for evaluating:

- performance
- efficiency
Application Main (AM) Agent
- first agent to execute on the client
- sets up the application once downloaded
- spawns other agents and creates artifacts
Application Main (AM) Agent
- first agent to execute on the client
- sets up the application once downloaded
- spawns other agents and creates artifacts

Service Finder (SF) Agent
- allows communication with type B service
- so as to find a list of type A services
- and concurrently download search agents
- also periodically monitoring for new type A services
A Sample Agent-Oriented Web 2.0 App - Design/1

- **Application Main (AM) Agent**
  - first agent to execute on the client
  - sets up the application once downloaded
  - spawns other agents and creates artifacts

- **Service Finder (SF) Agent**
  - allows communication with type B service
  - so as to find a list of type A services
  - and concurrently download search agents
  - also periodically monitoring for new type A services

- **Product Finder (PF) Agent**
  - allows communication with type A service
  - so as to get a list of available products
  - filtering them by user-defined parameters
  - also periodically monitoring for new products
Product Directory (PD) Artifact
- stores product data found by the PF agents
- filtering and ordering to build a list of best products
- updating results on the web page
A Sample Agent-Oriented Web 2.0 App - Design/2

- **Product Directory (PD) Artifact**
  - stores product data found by the PF agents
  - filtering and ordering to build a list of best products
  - updating results on the web page

- **Pausing (P) Artifact**
  - coordinates agents in the workspace
  - interruption/restarting of the research activities
A Sample Agent-Oriented Web 2.0 App - Design/2

- **Product Directory (PD) Artifact**
  - stores product data found by the PF agents
  - filtering and ordering to build a list of best products
  - updating results on the web page

- **Pausing (P) Artifact**
  - coordinates agents in the workspace
  - interruption/restarting of the research activities

- The architecture also include:
  - a single instance of **Page Artifact**
  - an instance of **HTTP Channel Artifact** for each PF Agent and one for the SF Agent.
Figura: The architecture of the sample client-side Web application in terms of agent, artifacts, and their interactions.
**Figura:** A screenshot of the implemented and executed application while it is running inside a Web browser.
A&A abstractions fit quite well the problem of representing entities of client-side web applications.
A&A abstractions fit quite well the problem of representing entities of client-side web applications.

Compared to mainstream approaches (JavaScript and AJAX), the model promote engineering properties such as

- modularity
- encapsulation
- separation of concerns
- maintainability
- extendibility
- reusability
A&A abstractions fit quite well the problem of representing entities of client-side web applications.

Compared to mainstream approaches (JavaScript and AJAX), the model promotes engineering properties such as:

- modularity
- encapsulation
- separation of concerns
- maintainability
- extendibility
- reusability

Useful to design and implement Web applications avoiding intricacies of low-level interfering mechanisms.
Limits and Future Works

Actually simpA-Web is just a possible implementation of the proposed agent-oriented model for client-side Web application

- others solutions can be developed
  - using natively agent-oriented language
Limits and Future Works

Actually simpA-Web is just a possible implementation of the proposed agent-oriented model for client-side Web application

- others solutions can be developed
  - using natively agent-oriented language
- lack of performance and integration difficulties should not be considered as diminishing the value of the programming model
  - possible future extensions of Web browser functionalities to support A&A abstractions
Limits and Future Works

Actually simpA-Web is just a possible implementation of the proposed agent-oriented model for client-side Web application

- others solutions can be developed
  - using natively agent-oriented language
- lack of performance and integration difficulties should not be considered as diminishing the value of the programming model
  - possible future extensions of Web browser functionalities to support A&A abstractions
- ongoing extensions of the basic model
  - to support interaction with web services
  - to design and develop also the server side of complex Web applications using A&A model.
Limits and Future Works

Actually simpA-Web is just a possible implementation of the proposed agent-oriented model for client-side Web application.

- Others solutions can be developed
  - Using natively agent-oriented language
- Lack of performance and integration difficulties should not be considered as diminishing the value of the programming model
  - Possible future extensions of Web browser functionalities to support A&A abstractions
- Ongoing extensions of the basic model
  - To support interaction with web services
  - To design and develop also the server side of complex Web applications using A&A model.
- The use of agents sets the stage for a possible evolution of Web 2.0 applications into *Semantic Web* applications
  - Intelligent agents as basic entities in Semantic Web
Thank you for your attention