Who should take care of the personalization?

Jarogniew Rykowski
The Poznan University of Economics, Department of Information Technology
Mansfelda 4, 60-854 Poznan, Poland
rykowski@kti.ae.poznan.pl

Abstract. In this paper, a new approach is proposed for the personalization of the access to different information sources (servers, Web pages, services, etc.) distributed across the Internet. In contrast to the traditional approaches, with personalization software owned and controlled by the owner of the information source, we propose to use the software that is prepared and controlled directly by the end-users. Our approach is based on software agent technology and imperative programming of agent code. The main goal of using personalization agents is twofold. First, such agents act as information brokers, adjusting both contents and format of the information to individual user requirements, restrictions imposed by the end-user hardware and communication means, current situation, etc. Second, the agents are used as autonomous monitors, individually informing agent owners about "important" (from the particular owner point of view) information changes. The fact of using source-independent personalization agents makes it possible to personalize access to such traditionally closed and fixed (i.e., unmanageable from the end-user point of view) information sources and systems, as e-banks, public Web portals and information servers, etc. Due to the fact the agents are prepared by (or at least for) particular users, the expenses related with the development of the agent code are in the major part incurred by these users.

1 Introduction

Rapid development of new information and communication technologies (ICT) introduced a new market of information services. Such a service is usually realized in a client-server mode, with information hosted at a certain location, and remotely operated clients. Recently, particular stress is put on mobile clients, i.e., clients with radio-connected devices of different type, purpose, and possibilities.

Mass introduction of new ICT services emphasized certain problems, mainly related with the well-known conflict between mass usage of these services, and

individual requirements and expectations of different service users. On one hand, a service is optimized from the point of view of the service owner, with a clear business goal – reducing service costs with keeping maximum efficiency, throughput, etc. On the other hand, each service client is different and obviously prefers his/her own, individual access mode to the information, taking into account not only human-related interests and habits, but also such user-independent factors, as communication costs, technical restrictions of end-user devices, current situation, geographical position, etc. So far, it looks like these are the service owners who are winning in the above-presented conflict, imposing several access conditions on the service clients. As a consequence, service personalization is usually quite restricted, as this is not a primary business goal for the service owner.

A question arises – what should be undertaken by a typical client in order to achieve a reasonable level of personalization? The response is twofold. Obviously, the client needs an efficient access to the information; however, these are the service owners who control such access, and such control cannot be passed by. On the other hand, certain information processing is needed, according to individual user requirements. This implies some user-related software, to be executed over service-related data. Here, the second question arises: where to execute such personalization software and who is in charge to start such execution? There are three basic possibilities. First, server-side execution – theoretically it looks like the optimum choice, however, service owners are not usually interested in providing such functionality. Second, client-side processing – the main restriction lies in the fact that end-user devices, especially mobile devices such as phones and palmtops, are not powerful enough to deal with complex data processing. Moreover, client-side processing requires certain communication bandwidth, usually costly and resource-consuming (note at least battery consumption, as well as a need for continuous on-line network access). And finally, the personalization software may be executed somewhere in the network, at a selected host. However, in this case we deal with the problem of a generic, distributed software environment for user-defined programs, to not say about certain security risks and the loss of privacy.

To solve the above problems, in this paper we propose a new strategy of user-defined personalization, both theoretical model, and the practical implementation. The strategy is based on imperatively programmed software agents, and a specialized application of the Agent Computing Environment (ACE) framework. Within this framework, the users are able to execute their own agents, choosing not only their own algorithm of the information processing, but also the execution place and time. Our approach makes it possible to personalize access to information sources of different type, purpose, characterized by different access methods and interfaces.

The remaining of the paper is organized as follows. First, in Section 2, we discuss main limitations of the current approaches to server-side and client-side personalization, especially those related with the software agent technology. In Section 3, a generic architecture of our agent-based personalization framework is described. In Section 4, we discuss some implementation issues, mainly security and privacy aspects, and basic methods for defining and executing user-owned

personalization software. Finally, in Section 5 we provide some final conclusions, and we point out some directions for the future work.

2 Current approaches to mass personalization

So far, software agents have not been widely used for mass personalization of the access to distributed information sources. Due to this fact, we are able to provide here only a limited state-of-the-art description related with sub-functionality of some agent-based and traditional distributed systems, the following features included: (1) personalization of distributed systems and information servers, (2) different strategies for monitoring and alerting, (3) human-to-agent communication methods, and (4) different programming methods for user-defined agents.

Usually, *personalization in distributed sy*stems, servers, and services is based on user-related data only – mainly client-side cookies and server-side user profiles. User profiles are either statically declared by the user (e.g., Yahoo and similar systems), or determined dynamically, taking into account current user activities (e.g., Click-stream Analysis, Web Usage Mining Systems, Collaborative Filtering). Some profiles may be fixed by analyzing individual usage of cookies [3]. There are three main disadvantages of such profile-based personalization: (1) personalization algorithm is fixed by the server owner, and thus it cannot be adapted to the specificity of the end-users, (2) profiles operate with limited number of parameters, usually quite restricted in type and scope, and (3) number of profiles is usually limited, to simplify profile management. However, higher is the level of profile generalization, lower is the satisfaction level of the end-users [7].

Recently, some attempts were proposed to provide user-defined personalization algorithms. For example, designers of the Web Services Experience Language WSXL introduced user description of service interfaces. Another proposal, XUL, a part of the Mozilla project [5], makes it possible to mix XML, CSS, RDF, and JavaScript technologies to personalize contents and formatting of Web pages. However, the above-mentioned personalization frameworks are still at the scientific-prototype stage, and their functionality is usually restricted to providing personalized Web pages being entry points to real Web pages and servers. What we propose is to use active, autonomous software agents rather than passive Web pages, to enable unrestricted personalization of any information source.

Monitoring and alerting (e.g., sending a message once something happens) by the use of software agents has been implemented in many systems, such as BargainFinder, NewsPage, and PumaTech's Mind-It [14]. Basic functionality of such systems is to facilitate e-shopping and searching for news (sport, stocks, weather forecast, etc.). There are two basic approaches to implement a monitoring system: client-side software, installed by the end-users and periodically polling specific Web servers and/or pages, and server-side agents, being internal parts of specialized information servers. In both cases, as the agents are prepared by software companies rather than the end-users, it is not possible to re-program detailed agent behavior, change parameter list (including the number and types of monitoring data sources), update execution schedule, etc. In the case of software-side monitoring agents, service owners are not interested in individualization of agent behavior, due

to pure economical reasons. Moreover, as one cannot foresee all the possible expectations of all the possible users, it is not feasible to provide generic monitoring software. As a consequence, types and locations of the monitored information sources are usually fixed, e.g., a sport-news agent informs users about sporting events; however, it is not possible to use this agent to distribute political news, even if the monitoring and alerting algorithms are quite similar in both cases.

In the case of client-side agents, periodical polling for information changes requires certain network bandwidth that maybe costly or even impossible to achieve in some cases (e.g., mobile phones). Note also, that even if the monitoring software is installed at the client-side, the end-users have limited possibilities to individualize the behavior of this software. Moreover, the monitoring process may be easily reversed, to monitor user and/or user software activities (e.g., spyware, however, some more useful functions as well, such as automatic updates). Software agent technology, being the core of our approach, is much better suitable for both server- and client-side monitoring, mainly due to the fact that these are the users who are responsible for preparation, and further management and execution of their agents.

Recently, some systems have been proposed using agents as brokers to Web Services, e.g., SEMOA [13] platform, and our Virtual Web Services [15]. Note, however, that Web Services are usually equipped with a stable, well defined interface and service ontology, while the case of a distributed system with heterogonous, dynamically evolving information sources is much more general. Thus, the personalization agents must be more flexible and user-oriented, forcing agent code and execution place/time to be defined directly by the end-users.

So far, standard WWW interface was preferred as a basic method *of human-to-machine communication*. We think that more stress should be put to natural-language interfaces via telecommunication means, such as SMS/MMS, e-mail, voice gateways, and Push-To-Talk PTT messaging, traditionally related with mobile phones and human-to-human communication [6]. Taking into account hardware/software limitations of the mobile phones, we think that the chatterbot conversation [10] looks like a good candidate for a primary agent interface [18]. So far, chatterbot interfaces were not widely used, especially in the context of the software agent technology. Such prototype systems as AMASE [2] and Microsoft Agent applications use only standard, pre-programmed communication chatterbots, with fixed set of keywords and possible actions, making the personalization practically impossible. What we propose is to extend the chatterbot interface by the possibility of applying user-defined agents with individual conversation algorithms, keywords, given information sources, etc.

As for the latter above-mentioned feature – programming method for remotely executed software agents, we may choose between a skeleton-based approach, declarative approach, and imperative programming. *Skeleton-based approach* is based on some predefined pieces of code being “patterns” for automatic creation of the agents to be executed in the name of the agent owners [1]. This approach maybe used in the case the overall system security is much more important than user privacy and code individualization. Usually, the agent “owner” is in the power to choose the place and time of execution of the software only, while the software

functionality is determined in advance by system designers. Even if this approach is widely used for such closed and fixed application areas as internet shops and marketplaces (e.g., Concordia agents), we think that it cannot be applied for user-defined personalization activities and ad-hoc cooperation.

The second above-mentioned programming technique – declarative programming – is usually based on a specialized programming language [8], originated from logical programming (Shoham's Agent0 programming language), artificial intelligence, goal-oriented programming (e.g., database programming languages) [4], etc. Even if declarative programming is much more flexible than using program skeletons, we think that it still cannot be applied for unrestricted user-defined personalization activities. First, similar to the skeleton-based technique, all the possible declarations (and thus total system functionality) must be known in advance. Each user has a choice of using or not certain declarations, however, he/she is not able to fully program details of the agent behavior [17]. Second, providing run-time "compilation" of a declaratively-programmed agent limits overall efficiency, both from the system (less throughput, more resources consumed – memory, CPU, etc.) and the agent owner (slow execution time) points of view.

Another approach to define software agents consists in the use of a classical, imperative programming language, such as Java. AMASE [2] is an agent-based system with java agents working in different mobile environments. Voyager is another example of usage of Java-based agents, with many successful e-commerce and e-business applications. Recently introduced Cougaar system uses agents composed by users with agent parts chosen from a set of well-defined, stable plugins encoded by system designers.

The most important problem of the Java-based agents is to achieve reasonable level of global system safety. Executing external Java code means, from a local system point of view, executing alien code, unknown and potentially dangerous. Even if the level of security provided by Java is considered as to be high, and several additional mechanisms are used – ciphering, digital signatures, anti-virus checkers – one cannot be sure the just executed code behaves well. Maybe in several cases this is more psychological than real menace; however, Java-based agents usually used in closed, mutually trusted environments. Moreover, in most of the cases, the users are not directly allowed to prepare the Java code. Instead, system-defined code is used as a set of "black boxes" (e.g., Voyager applications and Cougaar plugins).

3 Architecture of the APE personalization framework

To solve all the problems mentioned in Section 2, we propose to use one single technology – software agents. In our approach, called Agent-Based Personalization Environment (APE) we define software agents in the classical way, as presented in [9, 20]. A software agent is a program, executed at a given place, characterized by: (1) autonomy – agents process their work independently without the need for human management, (2) communication – agents are able to communicate with one another, as well as with humans, and (3) learning – agents are able to learn as they react with their environment and other agents or humans. As follows from the above definition, an agent may be programmed by its owner, thus allowing unrestricted

personalization of behavior of this agent [19]. Agents may be executed in different places [11], according to owners' needs and possibilities of the end-user hardware [12]. In particular, agents may continuously monitor given information sources and inform about detected information changes [16].

APE agents are individually programmed by (or at least for) selected users. Agent functionality, i.e., personalization scope, is under exclusive control of the agent owner. This is up to the user to decide about amount of work (and costs) for agent preparation, distribution, and execution. The user is able to determine the personalization algorithm (i.e., the agent code), distribution and execution strategy (time, place, and conditions for running the agents), and finally – human-to-agent communication interface. As the agents are prepared individually for given users, there is no need for global agent management, a global schema, a uniform communication protocol, user groups, profiles, etc. Instead, users are free to define the most adequate (for them) agent behavior and variables.

APE agents may be distributed across the network. In particular, agents may be executed in user private environment (home PC, a notebook, a portable, or even a mobile phone), at server-side near the information sources, and at a selected host (so called network-side). Agent distribution may relieve the system of huge data transfer, traditionally related with client-server operating mode, by shifting the agents close to the information sources. Moreover, agent mobility makes it possible to take profits of the personalization in the case the user is not equipped with a powerful end-user device, e.g., in a mobile environment.

In contrast to traditional approach of accessing information sources, APE agents are able to access the source in two modes: synchronously and asynchronously. In the *synchronous mode* (being a counterpart of a classical access), this is the agent owner who sends a request to the information source/service, using his/her agents as brokers. The agent is responsible for contacting the information source(s) and collecting the response. The just-collected information is prepared (contents) and formatted (both a form and a layout) according to user needs, hardware/software environment, and communication means (a bandwidth, costs, speed, etc.). Thus, APE agents act as information brokers, hiding some details related with access methods to the information sources, and personalizing the information obtained. In the *asynchronous mode*, APE agents act as autonomous monitors, operating in a selected network host, and informing their owners about "interesting" information changes. What is "interesting" is programmed by the user in the code of his/her agent(s). The main advantage of the asynchronous mode is the fact that the agent owner is bother by really important alerts, however, important only for him/her. As long as there is nothing "interesting" related with the information observed, there is no need for additional polling, verification, etc. Note that the alerts may be sent to non-advances devices, such as mobile phones (SMS/MMS messaging), or even to a traditional phone via a voice gateway (speech synthesis).

Beside the monitoring and formatting functionality, APE agents may be used for bonding originally independent information sources and services into one single, consistent (from the agent owner point of view, however) conglomerate. Such operation, so called "orchestration" of servers/services, needs no permission of the

information/service owner. As the orchestration is performed for a single user, user privacy is preserved. However, if needed, the above process may be related with a group of users and common agents, including ad-hoc, informal cooperation (e.g., tourist groups, people at certain geographical locations, etc.).

APE agents, to some extent, are able to provide a personification of the information sources/servers/devices they are related with. The agent owner, accessing his/her agents via a standard communication channel (e.g., a voice gateway, SMS/MMS messaging, e-mail) in a semi-natural language (cf. Section 2.4) has an impression of interacting with another human [18]. For example, we may consider an agent for an intelligent building. Agent owner, stuck in a traffic jam, may call the agent and ask to record a TV show. The agent searches for the show details in the Internet schedule, and starts recording with the VCR/DVD device. Note the agent owner contacts the system exactly in the same way as the family members, with all the technical details completely hidden. Note also, that the recording device does not have to be "intelligent", unless the agent is equipped with such "intelligence" to start/stop the device. However, such agent "intelligence" is quite simple (chatterbot conversation, a connection with an Internet search engine, a remote-control hardware link to the device, e.g., IrDA-operated), and may be realized with limited efforts.

Once developed and registered by an agent owner, the APE agents may be located and further executed at a selected place of the APE network of hosts. We assume that there are three basic classes of the hosts an APE agent may be sent to and executed: private hosts, generic network hosts, and server-side hosts. According to these host classes, we distinguish three basic agent pools: client-side pool, composed of the hosts controlled by the agent owners (i.e., ordinary end-users), middle-side pool, composed by some general-usage hosts, and source-side pool, composed by the hosts controlled by the service owners (i.e., the users offering some services and access to the information). The pools are characterized by different methods for migrating, storing, searching for, and executing the agents. Below, a general characteristic is given of each pool, together with a description of purpose and functionality of sample agents belonging to these pools.

A functionality of a host from the *source-side pool* is optimized towards reliable and efficient access to selected data sources, from the point of view of the information owner. Agents operating in source-side hosts are usually owned by the information owner. For security reasons, storing and executing "alien" agents belonging to the end-users is substantially limited. A typical source-side host is reduced to a set of gateways, able to standardize an access to the data source(s) connected, with limited support for public telecommunication facilities (WWW access, SMS/MMS/e-mail asynchronous messaging, etc.). The gateways are equipped with several mechanisms supporting efficient, parallel, multi-user access to the data sources, for example cache memories, proxies, synchronizers, locks, query optimizers and serializers, etc.

Accessing agents from source-side hosts is similar to accessing public Web servers and services. The difference is the agents provide some additional communication, wrapping and brokering functionality, requested by the users, as well as some uniformity of the external access to several information sources. However, nevertheless the end-users have limited control over source-side agents –

usually such agents are used as "black boxes", with limited possibilities of individualization of their behavior as well as the mode of operation.

Hosts from the *middle-side pool* are located in arbitrary chosen parts of the global network. In contrast to the source-side pool, middle-side hosts store and execute agents belonging to different users. A typical task list for the user agents covers: brokering among source-side and private agents, wrapping and formatting messages exchanged by the population of agents, providing access via different telecommunication means and protocols, etc. A stress is put on efficient access to the agents by the humans, using popular telecommunication channels and standards (WWW/WAP, SMS/MMS, e-mail, etc.). Agents from the middle-side pool are usually devoted to the tasks related with network-side monitoring – detecting information changes that are "interesting" for the agent owners. As already stated, what is "interesting" is programmed by the agent owner in the agent code.

Architecture and usage of a host from the *client-side pool* strongly depends on technical and communicational possibilities of an end-user hardware/software the agent owner posses at the moment. Private agents may be executed for example in the scope of a stationary PC, mobile equipment (a PDA, a notebook, or even an intelligent mobile phone). It is up to the agent owner to locate his/her agents either in a host from the middle-side pool, or in the private (i.e., client-side) host. In the first case, the network traffic may be substantially reduced, however, remotely executed user agents are less secured (from the user point of view) and less efficient (mainly because of additional security checks, cf. Section 4). In the second case, all the user agents are executed in a trusted (still, only from the agent owner point of view) environment, however, a lot of data must be transferred among distributed hosts.

For the agents executed at a portable/mobile device, a stress is put on fast and user-friendly human-to-agent communication. The technical capabilities of the device strongly limit the possibilities of executing the agents (small memory, limited battery time, difficult management, etc.). Thus, usually only a few private agents are located in a mobile host capable of performing some simple tasks, e.g., formatting of an alert message, filtering incoming messages, generating sound alerts, etc.

APE framework provides a possibility of defining and using several specialized agents called input/output gateways, able to communicate with the external world (including both software and humans) via communication channels of different type and purpose. Number and types of the gateways used (including some specific parameters, as a phone number for an SMS center, an address for a SMTP/POP3 server, etc.) is local-administrator dependent. Note that the gateways are implemented as agents, thus one may easily extend the framework by some specific communication channels, for example a dedicated application for contacting and programming agents, file system/NFS gateway able to exchange information via common files, etc.

In general, two basic types of communication channels are available: textual and Web-based. A *textual channel* is able to exchange flat (unformatted) text messages, usually among humans and agents. Physically, textual channels may use such media as an e-mail SMPT/POP3 connection, SMS (Short Message System)/MMS connection with a telecommunication network, a voice gateway, etc. Once sent by a

textual message, an agent acts as a chatterbot, analyzing the message via keyword extraction and analysis. The semi-natural access to an agent in a chatterbot manner is especially useful for non-advanced users, as well as for users temporary handicapped due to limited hardware possibilities and communication costs. For example, an SMS message may be used to check the most important information during a journey, once a stationary PC is further used to get the complex information while the user is back home. *Web-based channels* are used to access an agent via a WWW/WAP page, and from specialized applications. These channels use personal, semi-automatic formatting of both contexts and presentation of the data to be sent. To this goal, XSL-T technology was adapted with XSL transformations defined in a personal manner and stored in private agent variables. In a case of a conversation with a human, automatic detection of end-user device may be used, thus restricting the communication. For example, a small textual message is sent to a mobile phone using WAP connection, similar message with the same contents however some additional formatting is sent to a PDA device, and full text&graphic message is sent to a stationary PC.

4 Implementation issues

The APE idea was implemented using the Agent Computing Environment ACE, the framework originally developed by us for supporting owners of mobile phones in accessing the Internet servers [16]. The framework is based on a set of distributed Agent Servers [17], each of them capable of storing and executing software agents. The agents are imperatively programmed by the use of certain programming languages, both standard and dedicated only for ACE framework. The agents may be moved among Agent Servers. Agent Servers may be located in both stationary and mobile devices. In case of stationary equipment, multi-user, multi-agent, mass-usage Agent Servers may be used. In case of mobile devices, characterized by limited hardware and software possibilities and high communication costs, personal, single-agent, light-weight Agent Servers may be used. Depending on hardware and communication restrictions, and current situation, a user has a choice in determining a place of execution of agents.

Due to the restrictions of the skeleton-based and declarative programming techniques (cf. Section 2.3), we propose to use imperative programming for setting up agent behavior, directly by the agent owners. In order to provide reasonable level of portability (migration) of the agent, and reasonable level of overall system security, we propose to apply two primary programming techniques: interpretation connected with run-time code inspection for “untrusted” agents, and compilation for the “trusted” ones.

The main problem related with unrestricted usage of imperative code concerns limited system security. Remotely executed, imperatively programmed agents are treated as the “alien” code, potentially dangerous for the local environment. Such anxiety may be justified by insufficient level of code verification, or simply by pure psychological reaction of local system administrators. Even if from the “technical” point of view several security mechanisms are applied for the external code verification (i.e., code encryption and signing by digital certificates, built-in security

verification for the compilers and kernels of the operating systems, etc.), the psychological fear maybe a serious obstacle for wide acceptance of user-defined, not-known in advance agents. However, similar problem has been already successfully resolved in the domain of the operating systems, by introducing two basic programming techniques: a shell language, used for example for preparing batch programs and desktop icons, and different programming languages, used for design of the application programs, further compiled to "executables". Shell scripts are usually simple programs, in contrast to application programs – usually quite complicated and compiled (installed) prior to the execution. Compiled "executables" are used by the ordinary users as "black boxes", with limited parameterization and no possibility of re-programming internal functionality. Most operating-system users are entitled only to manipulate shell scripts, and only some (usually system administrators) are able to install and control executables.

Applying this approach to the APE agents, we propose two basic techniques of setting up safe and secured activity behavior: the dedicated shell language, and full compilation. The shell language is used for programming mobile, remotely executed, user-defined activities. The language is based on the XML standard, and its syntax computational power is similar to the widely known shell programming languages. Note that we were not able to adapt any existing XML query language, as well as any generic XML transformation language, as these languages are specialized for node processing, generating a set of XML nodes as a result of processing of queries/other nodes. Instead, we adopted typical shell syntax, adjusting it to the framework of the XML documents. In the proposed agent-shell programming language, the following shell-language statements may be used: *variable statement* (variable definition), *procedure definition* (similar to procedure/method definition in most of the imperative programming languages), *if-then-else* choice and *while* loops with conditional statements, and procedure *call* with *return* statement.

Due to the fact that the XML-programmed agents are treated as an "alien" code (from the point of view of the owner/administration of the agent execution environment), we introduce additional run-time checking: verification of the maximum agent execution time, and maximum space (quota) for temporal variables. Such checking is performed prior to the execution of each program line (XML node), by a comparison of granted and consumed amount of resources (mainly total CPU time and memory load). Badly-behaving agents are detected and their execution is stopped. As a result, it is not possible for an agent to loop (intentionally or not) forever and to consume too much memory/disk space, slowing down or even blocking other agents. To our best knowledge, there is no similar run-time verification for any shell-like programming language.

On the contrary, agents prepared by "trusted" (still, from the local environment point of view) users are written in Java, being an efficient and portable programming language. There is no additional run-time checking for such agents, except for standard security procedures built-in into the local Java Virtual Machine (memory consumption and quota, verification of the access rights, especially those related with accessing external software, digital signing and verification of the program code, etc.). Note that, even if the Java-based agents cannot be developed by ordinary users,

such agents may be used as "black boxes", similar to the applications of a typical operating system. Note also that the Java-based agents may be used as brokers to external resources and software, including non-standard communication means (public SMS/MMS/e-mail channels, WWW-based access via individualized pages, etc.) [16].

Typically, we assume that most of the complex tasks are realized as Java-based agents, with the code prepared by system designers. The shell-based agents are used for the personalization purposes: linking, formatting, and presenting information obtained from different places and in different form, monitoring and alerting, adjusting the results to the individual requirements of the agent owner, as well as to hardware/software/communication limitations, etc. This is in turn similar to a typical operating system, with shell scripts and desktop icons personalizing the usage of the system-controlled applications. The local environment (and other agents) is secured enough, and, as the majority of the complex and resource-consuming tasks is realized by Java-based agents, the whole system is fast, both from the agent owner (small response time), and from the system point of view (large throughput).

5 Conclusions

In contrast to current solutions, the proposed personalization strategy is characterized by several advantages. First, the costs related with the preparation and the execution of the personalization software are incurred by the end-users. Each user is able to define his/her individual personalization scope and possible expenses. Second, it is possible to personalize access to traditionally closed and fixed (from the user point of view) information sources, such as e-banks and public Web portals. Third, it is up to the user to choose the information source(s), data processing algorithm, information scope, date and time of software execution, etc. Fourth, personalized access may be enriched by individual monitoring of "important" information changes and asynchronous alerting if "something interesting" happens with the monitored information. What is "interesting" for a particular user is defined by him/her in his/her personalization software. And fifth, it is possible to utilize several communication channels of different type, purpose, bandwidth, costs, etc., even those not necessary related with the information source, such as SMS/MMS/e-mail messaging for Web servers.

The system is flexible and open for new services, communication standards, users, etc. Due to the brokerage of public agents, the new services and protocols may be added in an invisible (for an ordinary user) way. Existing applications and distributed systems may profit from using software agents as monitors and personalizers [18]. Mobile agents and APE/ACE applications, apart from personalization of an access to distributed resources, are able to take some benefits of modern communication channels, such as WAP/WML, SMS/MMS, and PTT (Push-To-Talk)/voice access. To our best knowledge, there is not a single proposal up to now to use imperative, mobile, user-defined software agents for personalization of a distributed environment that is directly comparable with our approach.

Potential application areas of the APE/ACE framework are the following: advanced and individual controlling of database access, personal monitoring,

asynchronous notification for changes, personalization of closed systems, mobile access to databases, enabling access for non-advanced and handicapped users, mobile applications, etc.

The APE/ACE framework was implemented and tested as two industrial applications: a universal information system for users of mobile phones, and as personal monitoring software for an internet bank. Due to the lack of space, we are not able to provide here detailed results of the measurement of the system efficiency. We mention only that we measured average system response time and agent execution time for a population of up to 100000 of agents, created artificially as multiply clones of agents developed and used by a hundred of real human users. The obtained results – system response time counted in parts of a second for SMS, e-mail and WWW-based gateways, average agent execution time up to 80 ms, and system throughput up to 30 agent executions per second under maximum load for 1000 hours (more than a month) of continuous test – proved the whole system is fast and efficient, especially for "handicapped" owners of mobile phones during an access to distributed Internet information sources.

References

1. AgentLand, *home page*, http://www.agentland.com/, (AgentLand, Cybion, Paris, France, 2006).

2. R. Pascotto, AMASE: Agent-based Mobile Access to Information Services, ACTS project homepage, in *CORDIS, ACTS – Advanced Communication Technologies and Services* (European Commission, Dir INFSO, ACTS Central Office, Brussels, 2002), http://cordis.europa.eu/infowin/acts/analysys/products/thematic/agents/ch3/amase.htm.

3. M. Bonett, M., Personalization of Web Services: Opportunities and Challenges, *Ariadne* Issue 28 (June 2001), http://www.ariadne.ac.uk/issue28/personalization/intro.html.

4. DMAL, *DARPA Agent Markup Language Homepage* (DARPA, Arlington, VA, 2006), http://www.daml.org/.

5. N. Deakin, *XUL Tutorial* (XUL Planet, Sympatico, CA, last updated 19 February 2006), http://www.xulplanet.com/tutorials/xultu/.

6. eMobile, *SMS Software Solutions* (eMobile, Singapore, 2005), http://www.emobile.com.sg.

7. P. Farjani, C. Gorg and F. Bell, A Mobile Agent-Based Approach for the UMTS/VHE Concept, ACTS project homepage, in *CORDIS, ACTS – Advanced Communication Technologies and Services* (European Commission, Dir INFSO, ACTS Central Office, Brussels, 2004), http://cordis.europa.eu/infowin/acts/analysys/products/thematic/agents/ch3/cameleon.htm.

8. The Foundation for Intelligent Physical Agents, *Welcome to FIPA* (IEEE Foundation for Intelligent Physical Agents, Piscataway, NJ, 2006), http://www.fipa.org/.

9. S. Franklin and A. Graesser, Is it an Agent, or just a Program? A Taxonomy for Autonomous Agents, in: *Lecture Notes in Artificial Intelligene, Volume 1193: Proceedings of the 3rd International Workshop on Agent Theories, Architectures, and Languages* (Springer-Verlag, Berlin, Heidelberg,1996), pp. 21-35.

10. D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition* (Prentice-Hall, Upper Saddle River, NJ, 2000).

11. D. Kotz and R.S. Gray, Mobile agents and the future of the Internet, *ACM Operating Systems Review* **33**(3), 7-13 (1999).

12. D. Milojicic, Trend Wars – mobile agent applications, *IEEE Concurrency* **7**(3), 80-90 (July-September 1999).

13. U. Pinsdorf, J.Peters, M.Hoffmann and P. Gupta, Context-Aware Services based on Secure Mobile Agents, in: *Proceedings of 10th International Conference on Software, Telecommunications and Computer Networks, SoftCOM 2002* (IEEE Communications Society, University of Split, Croatia, 2002), pp. 366-370, http://www.semoa.org/docs/papers/pinsdorf2002c.pdf.

14. Intellisync, *Pumatech announces Mind-it software version 5.2* (Intellisync Corporation, San Jose, CA, 27 November 2001), http://www.intellisync.com/.

15. J. Rykowski and W. Cellary, Virtual Web Services - Application of Software Agents to Personalization of Web Services, in: *6th International Conference on Electronic Commerce ICEC 2004, Delft (The Netherlands)* (ACM Publishers, New York, 2004), pp. 409-418.

16. J. Rykowski and A. Juszkiewicz, Personalization of Information Delivery by the Use of Agents, in: *Proceedings of the IADIS International Conference WWW/Internet 2003, ICWI 2003, Algarve, Portugal, November 5-8, 2003*. (IADIS, Algarve, Portugal, 2003), pp. 1056-1059

17. J. Rykowski, Management of information changes by the use of software agents, *Cybernetics and Systems* **37**(2/3), 229-260 (March-May 2006)

18. J. Rykowski, Using software agents to personalize natural-language access to Internet services in a chatterbot manner, in: *2nd International. Conference Language And Technology L&T'05* (Adam Mickiewicz University Press, Poznan, Poland, 2005), pp. 269–273.

19. B. Schiemann, E. Kovacs and K.Röhrle, "Adaptive Mobile Access to Context-Aware Services", Proc. of the 3rd International Workshop on Mobile Agents, Palm Springs, USA, MA'99 (IEEE, Piscataway, NJ, 1999), pp. 190-203.

20. M. Wooldridge and N. R. Jennings, Intelligent agents: theory and practice, *Knowledge Engineering Review* **10**(2), 115-152 (1995).