

Challenges of Middleware for the Internet of Things

Michal Nagy, Artem Katasonov, Oleksiy Khriyenko, Sergiy Nikitin,
Michal Szydłowski and Vagan Terziyan
*University of Jyväskylä
Finland*

1. Introduction

Recent advances in networking, sensor and RFID technologies allow connecting various physical world objects to the IT infrastructure, which could, ultimately, enable realization of the Internet of Things and the Ubiquitous Computing visions. Also, this opens new horizons for industrial automation, i.e. automated monitoring, control, maintenance planning, etc, of industrial resources and processes. A much greater number of resources than that of today (machines, infrastructure elements, materials, products) can be connected to the IT systems, thus be monitored and potentially controlled. Such development will also necessarily create demand for a much wider integration with various external resources, such as data storages, information services, and algorithms, which can be found in other units of the same organization, in other organizations, or on the Internet.

The interconnectivity of computing and physical systems could, however, become “the nightmare of ubiquitous computing” (Kephart & Chess, 2003) in which human operators will be unable to manage the complexity of interactions in the system, neither even architects will be able to anticipate that complexity, and thus to design the system. The IBM vision of autonomic computing (Kephart & Chess, 2003) proclaims the need for computing systems capable of “running themselves” with minimal human management which is mainly limited to definition of some higher-level policies rather than direct administration. The computing systems will therefore be self-managed, which, according to the IBM vision, includes self-configuration, self-optimization, self-protection, and self-healing. The IBM vision emphasizes that the run-time self-manageability of a complex system requires its components to be to a certain degree autonomous themselves. Following this, we envision that the software agent technologies will play an important part in building such complex systems. Agent-based approach to software engineering is considered to be facilitating the design of complex systems (Jennings, 2001). In the multi-agent systems field, a significant amount of attention is paid to the task of building decentralized systems capable of supporting spontaneous configuration, tolerating partial failures, or arranging adaptive reorganization of the whole system (Mamei & Zambonelli, 2006).

A major problem is inherent heterogeneity in ubiquitous computing systems, with respect to the nature of components, standards, data formats, protocols, etc, which creates significant

obstacles for interoperability among the components. Semantic technologies are viewed today as a key technology to resolve the problems of interoperability and integration within heterogeneous world of ubiquitously interconnected objects and systems. Semantic technologies are claimed to be a qualitatively stronger approach to interoperability than contemporary standards-based approaches (Lassila, 2005). The Internet of Things should become in fact the Semantic Web of Things (Brock & Schuster, 2006). We subscribe to this view. Moreover, we apply semantic technologies not only to facilitate the discovery of heterogeneous components and data integration, but also for the behavioral control and coordination of those components (i.e. prescriptive specification of the expected behavior, declarative semantic programming).

It seems to be generally recognized that achieving the interoperability by imposing some rigid standards and making everyone comply could not be a case in ubiquitous environments. Therefore, the interoperability requires existence of some middleware to act as the glue joining heterogeneous components together. A consistent set of middleware, offering application programming interfaces, communications and other services to applications, will simplify the creation of applications and help to move from static programming approaches towards a configurable and dynamic composition capability (Buckley, 2006).

In this chapter, we describe our vision of such a middleware for the Internet of Things, which has also formed the basis for our research project UBIWARE. The project aims at a new generation middleware platform which will allow creation of self-managed complex systems, in particular industrial ones, consisting of distributed, heterogeneous, shared and reusable components of different nature, e.g. smart machines and devices, sensors, RFIDs, web-services, software applications, humans along with their interfaces, and others. Such middleware will enable various components to automatically discover each other and to configure a system with complex functionality based on the atomic functionalities of the components.

2. Semantic web technology for Internet of Things

2.1 Semantic Web technology for ubiquitous computing

According to (Buckley, 2006), the actual power of the Internet of Things arises from the fact that the devices are interconnected. Interoperability requires that clients of services know the features offered by service providers beforehand and semantic modeling should make it possible for service requestors to understand what the service providers have to offer. This is a key issue for moving towards an open-world approach, where new or modified devices and services may appear at any time, and towards device networks capable of dynamically adapting to context changes as may be imposed by application scenarios. This also has implications on requirements for middleware, as these are needed to interface between the devices that may be seen as services, and applications. Devices in the Internet of Things might need to be able to communicate with other devices anywhere in the world. This implies a need for a naming and addressing scheme, and means of search and discovery. The fact that devices may be related to an identity (through naming and addressing) raises in turn a number of privacy and security challenges. A consistent set of middleware, offering application programming interfaces, communications and other services to applications, will simplify the creation of services and applications. We need to move from

static programming approaches towards a configurable and dynamic composition capability.

In (Lassila & Adler, 2003), the ubiquitous computing is presented as an emerging paradigm qualitatively different from current personal computing scenarios by involving dozens and hundreds of devices (sensors, external input and output devices, remotely controlled appliances, etc). A vision was presented for a class of devices, so called "Semantic Gadgets", that will be able to combine functions of several portable devices users have today. Semantic Gadgets will be able to automatically configure themselves in new environments and to combine information and functionality from local and remote sources. Semantic Gadgets should be capable of semantic discovery and device coalition formation: the goal should be to accomplish discovery and configuration of new devices without "a human in the loop." Authors pointed out that a critical to the success of this idea is the existence or emergence of certain infrastructures, such as the World Wide Web as a ubiquitous source of information and services and the Semantic Web as a more machine- and automation-friendly form of the Web.

Later, (Lassila 2005a) and (Lassila, 2005b) discussed possible application of Semantic Web technologies to mobile and ubiquitous computing arguing that ubiquitous computing represents the ultimate "interoperability nightmare". This work is motivated by the need for better automation of user's tasks by improving the interoperability between systems, applications, and information. Ultimately, one of the most important components of the realization of the Semantic Web is "serendipitous interoperability", the ability of software systems to discover and utilize services they have not seen before, and that were not considered when and where the systems were designed. To realize this, qualitatively stronger means of representing service semantics are required, enabling fully automated discovery and invocation, and complete removal of unnecessary interaction with human users. Avoiding a priori commitments about how devices are to interact with one another will improve interoperability and will thus make dynamic ubiquitous computing scenarios without any choreographing more realistic. Semantic Web technologies are qualitatively stronger approach to interoperability than contemporary standards-based approaches.

2.2 Semantic Web technology for coordination

When it comes to developing complex, distributed software-based systems, the agent-based approach was advocated to be a well suited one (Jennings, 2001). From the implementation point of view, agents are a next step in the evolution of software engineering approaches and programming languages, the step following the trend towards increasing degrees of localization and encapsulation in the basic building blocks of the programming models (Jennings, 2000). After the structures, e.g., in C (localizing data), and objects, e.g., in C++ and Java (localizing, in addition, code, i.e. an entity's behavior), agents follow by localizing their purpose, the thread of control and action selection. An agent is commonly defined as an encapsulated computer system situated in some environment and capable of flexible, autonomous action in that environment in order to meet its design objectives (Wooldridge, 1997).

The problem of "crossing the boundary" from the domain (problem) world to the machine (solution) world is widely recognized as a major issue in software and systems engineering. Therefore, when it comes to designing software, the most powerful abstractions are those that minimize the semantic distance between the units of analysis that are intuitively used to

conceptualize the problem and the constructs present in the solution paradigm (Jennings, 2000). A possibility to have the same concept, i.e. agent, as the central one in both the problem analysis and the solution design and implementation can make it much easier to design a good solution and to handle the complexity. In contrast, e.g. the object-oriented approach has its conceptual basis determined by the underlying machine architecture, i.e. it is founded on implementation-level ontological primitives such as object, method, invocation, etc. Given that the early stages of software development are necessarily based on intentional concepts such as stakeholders, goals, plans, etc, there is an unavoidable gap that needs to be bridged. (Bresciani et al., 2004) even claimed that the agent-oriented programming paradigm is the only programming paradigm that can gracefully and seamlessly integrate the intentional models of early development phases with implementation and run-time phases. In a sense, agent-oriented approach postpones the transition from the domain concepts to the machine concepts until the stage of the design and implementation of individual agents (given that those are still to be implemented in an object-oriented programming language).

Although the flexibility of agent interactions has many advantages when it comes to engineering complex systems, the downside is that it leads to unpredictability in the run time system; as agents are autonomous, the patterns and the effects of their interactions are uncertain (Jennings, 2000). This raises a need for effective coordination, cooperation, and negotiation mechanism. (Those are in principle distinct, but the word “coordination” is often used as a general one encompassing all three; so for the sake of brevity we will use it like that too). (Jennings, 2000) discussed that it is common in specific systems and applications to circumvent these difficulties, i.e. to reduce the system’s unpredictability, by using interaction protocols whose properties can be formally analyzed, by adopting rigid and preset organizational structures, and/or by limiting the nature and the scope of the agent interplay. However, Jennings asserted that these restrictions also limit the power of the agent-based approach; thus, in order to realize its full potential some longer term solutions are required.

The available literature sketches two major directions of search for such a longer term solution:

- D1: Social level characterization of agent-based systems. E.g. (Jennings, 2000) stressed the need for a better understanding of the impact of sociality and organizational context on an individual’s behavior and of the symbiotic link between the behavior of the individual agents and that of the overall system.
- D2: Ontological approaches to coordination. E.g. (Tamma et al., 2005) asserted a need for common vocabulary for coordination, with a precise semantics, to enable agents to communicate their intentions with respect to future activities and resource utilization and get them to reason about coordination at run time. Also (Jennings et al., 1998) put as an issue to resolve the question about how to enable individual agents to represent and reason about the actions, plans, and knowledge of other agents to coordinate with them.

In our work, we attempt to provide a solution advancing into both D1 and D2 and somewhat integrating both. Some basic thinking, leading our work, follows.

3. The Vision and Approach

We believe that the ultimate goal is the vision of the Global Understanding Environment (GUN) (Terziyan, 2003; Terziyan, 2005; Kaykova et al., 2005a). We made the first step in the SmartResource project (2004-2006). Figure 1 depicts our research roadmap.

Global Understanding Environment (GUN) aims at making heterogeneous resources (physical, digital, and humans) web-accessible, proactive and cooperative. Three fundamentals of such platform are Interoperability, Automation and Integration. Interoperability in GUN requires utilization of Semantic Web standards, RDF-based metadata and ontologies and semantic adapters for the resources. Automation in GUN requires proactivity of resources based on applying the agent technologies. Integration in GUN requires ontology-based business process modeling and integration and multi-agent technologies for coordination of business processes over resources.

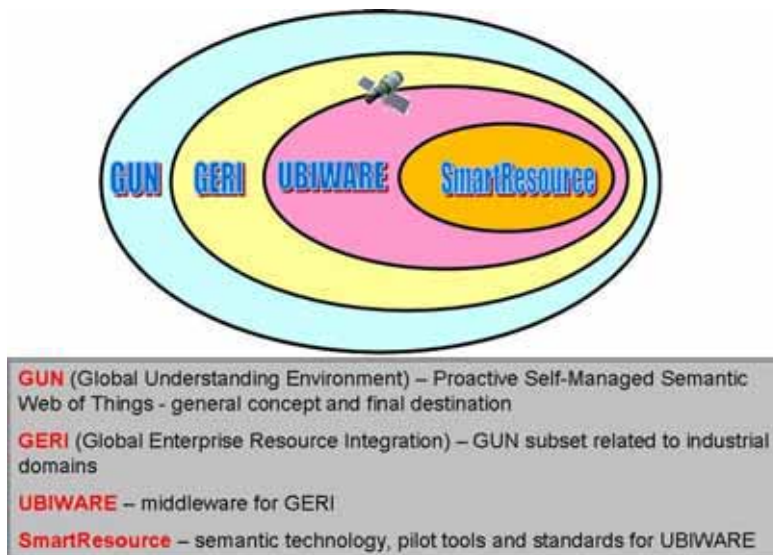


Fig. 1. The research roadmap towards GUN.

When applying Semantic Web in the domain of ubiquitous computing, it should be obvious that Semantic Web has to be able to describe resources not only as passive functional or non-functional entities, but also to describe their behavior (proactivity, communication, and coordination). In this sense, the word “global” in GUN has a double meaning. First, it implies that resources are able to communicate and cooperate globally, i.e. across the whole organization and beyond. Second, it implies a “global understanding”. This means that a resource A can understand all of (1) the properties and the state of a resource B, (2) the potential and actual behaviors of B, and (3) the business processes in which A and B, and maybe other resources, are jointly involved.

Main layers of GUN can be seen in Figure 2. Various resources can be linked to the Semantic Web-based environment via adapters (or interfaces), which include (if necessary) sensors with digital output, data structuring (e.g. XML) and semantic adapter components (XML to

Semantic Web). Software agents are to be assigned to each resource and are assumed to be able to monitor data coming from the adapter about the state of the resource, make decisions on the behalf of the resource, and to discover, request and utilize external help if needed. Agent technologies within GUN allow mobility of service components between various platforms, decentralized service discovery, FIPA communication protocols utilization, and multi-agent integration/composition of services.



Fig. 2. Main layers of GUN.

When applying the GUN vision, each traditional system component becomes an agent-driven “smart resource”, i.e. proactive and self-managing. This can also be recursive. For example, an interface of a system component can become a smart resource itself, i.e. it can have its own responsible agent, semantically adapted sensors and actuators, history, commitments with other resources, and self-monitoring, self-diagnostics and self-maintenance activities. This could guarantee high level of dynamism and flexibility of the interface. Such approach definitely has certain advantages when compared to other software technologies, which are integral parts of it, e.g. OOSE, SOA, Component-based SE, Agent-based SE, and Semantic SE. This approach is also applicable to various conceptual domain models. For example, domain ontology can be considered a smart resource, what would allow having multiple ontologies in the designed system and would enable their interoperability, on-the-fly mapping and maintenance, due to communication between corresponding agents.

In one sense, our intention is to apply the concepts of automatic discovery, selection, composition, orchestration, integration, invocation, execution monitoring, coordination, communication, negotiation, context awareness, etc (which were, so far, mostly related only

to the Semantic Web-Services domain) to a more general “Semantic Web of Things” domain. Also we want to expand this list by adding automatic self-management including (self-*) organization, diagnostics, forecasting, control, configuration, adaptation, tuning, maintenance, and learning.

According to a more global view to the Ubiquitous Computing technology:

- UBIWARE will classify and register various ubiquitous devices and link them with web resources, services, software and humans as business processes’ components;
- UBIWARE will consider sensors, sensor networks, embedded systems, alarm detectors, actuators, communication infrastructure, etc. as “smart objects” and will provide similar care to them as to other resources.

Utilization of the Semantic Web technology should allow:

- Reusable configuration patterns for ubiquitous resource adapters;
- Reusable semantic history blogs for all ubiquitous components;
- Reusable semantic behavior patterns for agents and processes descriptions;
- Reusable coordination, design, integration, composition and configuration patterns;
- Reusable decision-making patterns;
- Reusable interface patterns;
- Reusable security and privacy policies.

Utilization of the Distributed AI technology should allow:

- Proactivity and autonomic behavior;
- Communication, coordination, negotiation, contracting;
- Self-configuration and self-management;
- Learning based on live blog histories;
- Distributed data mining and knowledge discovery;
- Dynamic integration;
- Automated diagnostics and prediction;
- Model exchange and sharing.

4. The core of the middleware

4.1 Challenges

The main objectives of the Ubiware core (UbiCore) are the following. It has to give every resource a possibility to be smart (by connecting a software agent to it), in a sense that it would be able to proactively sense, monitor and control its own state, communicate with other components, compose and utilize own and external experiences and functionality for self-diagnostics and self-maintenance. It has to enable the resources to automatically discover each other and to configure a system with complex functionality based on the atomic functionalities of the resources. It has to ensure a predictable and systematic operation of the components and the system as a whole by enforcing that the smart resources act as prescribed by their organizational roles and by maintaining the “global” ontological understanding among the resources. The latter means that a resource A can understand all of (1) the properties and the state of a resource B, (2) the potential and actual behaviors of B, and (3) the business processes in which A and B, and maybe other resources, are jointly involved.

Several Agent Programming Languages (APLs) have been developed by researchers working in internal architectures and approaches to implementation of software agents. Examples of such languages are AGENT-0 (Shoham, 1993), AgentSpeak(L) (Rao, 1996), 3APL (Dastani et al., 2004) and ALPHA (Collier et al., 2005). All of those are declarative rule-based languages and are based on the first-order logic of n-ary predicates. All of them are also inspired by the Beliefs-Desires-Intentions architecture (Rao & Georgeff, 1991). For example, an agent program in ALPHA consists of declarations of the beliefs and goals of that agent and declaration of a set of rules, including belief rules (generating new beliefs based on existing ones), reactive rules (invoking some actions immediately) and commitment rules (adopting a commitment to invoke an action). Sensors (perceiving environment and generating new beliefs) and actuators (implementing the actions to be invoked) are then pieces of external code, namely in Java.

Based on the review of the languages mentioned above, we list the following important features of them, which should also be realized in UBIWARE's language for specification of behavior models:

- Ability to specify beliefs (something that the agent believes to be true) and goals (something that the agent does not believe to be true but wants to eventually become true).
- Ability to describe behavior rules, i.e. actions taken when a certain condition is met (can be either presence or absence of certain beliefs or presence of certain goals). Ability to have as the result of firing rule all of following:
 - adding/removing beliefs
 - engaging sensors and actuators
 - creating commitments (actions to be executed later)
- Ability to describe plans, i.e. predefined sequences of actions.
- Ability to describe commitments that are:
 - executed when certain condition is met
 - dropped when certain condition is met (or is not met anymore)

Agent-oriented approach postpones the transition from the domain concepts to the machine concepts until the stage of the design and implementation of individual agents. The advantage of using an APL is that the transition is postponed even further, until the implementation of particular sensors and actuators. This advantage seems to be, however, the only one that is considered. We did not encounter in literature approaches that would extend the role of APL code beyond the development stage. APL code is assumed to be written by the developer of an agent and either compiled into an executable program or interpreted in run-time but remaining an agent's intrinsic and static property. APL code is not assumed to ever come from outside of the agent in run-time, neither shared with other agents in any way.

Such export and sharing of APL code would, however, probably make sense. Methodologies for design of agent-based systems like OMNI (Vazquez-Salceda, 2005) describe an organizational role with a set of rules, and an APL is a rule-based language. So, using an APL for specifying a role sounds as a natural way to proceed. The difference is that APL code corresponding to a role should naturally be a property of and controlled by the organization, and accessed by the agents' enacting the role potentially even in the run-time. Run-time access would also enable the organization to update the role code if needed.

The second natural idea is that the agents may access a role's APL code not only in order to enact that role, but also in order to coordinate with the agents playing that role. As one

option, an agent can send to another agent a part of its APL code to communicate its intentions with respect to future activities (so there is no need for a separate content language). As another option, if a role's code is made public inside the organization, the agents may access it in order to understand how to interact with, or what to expect from, an agent playing that role.

However, when thinking about using the existing APLs in such a manner, there are at least two issues:

- The code in an APL is, roughly speaking, a text. However in complex systems, a description of a role may need to include a huge number of rules and also a great number of beliefs representing the knowledge needed for playing the role. Also, in a case of access of the code by agents that are not going to enact this role, it is likely that they may wish to receive only a relevant part of it, not the whole thing. Therefore, a more efficient, e.g. a database-centric, solution is probably required.
- When APL code is provided by an organization to an agent, or shared between agents, mutual understanding of the meaning of the code is obviously required. While using first-order logic as the basis for an APL assures understanding of the semantics of the rules, the meaning of predicates used in those rules still needs to be consistently understood by all the parties involved. On the other hand, we are unaware of tools allowing unambiguous description of the precise semantics of n-ary predicates.

As a solution to these two issues, we see creating an APL based on the W3C's Resource Description Framework (RDF). RDF uses binary predicates only, i.e. triples (n-ary predicates can be represented nevertheless, of course, using several approaches). For RDF, tools are available for efficient database storage and querying, and also for explicit description of semantics, e.g. using OWL. Our proposition for such an RDF-based APL is the Semantic Agent Programming Language (S-APL).

4.2 Proposed solution

The answer to the requirements mentioned above is our Semantic Agent Programming Language (S-APL). S-APL is built on top of these axioms:

- Everything is a belief. All other mental attitudes such as desires, goals, commitments, behavioral rules are just complex beliefs.
- Every belief is either a semantic statement (subject-predicate-object triple) or a linked set of such statements.
- Every belief has the context that restricts the scope of validity of that belief. Beliefs have any meaning only inside their respective contexts.
- Statements can be made about contexts, i.e. contexts may appear as subjects or/and objects of triples. Such statements give meaning to contexts. This also leads to a hierarchy of contexts (not necessarily a tree structure though).
- There is the general context G, which is the root of the hierarchy. G is the context for global beliefs (as opposed to local ones). Nevertheless, every local belief, through the hierarchical chain of its contexts, is linked to G.

The notation that is selected for use in S-APL is a subset of Notation3 (Berners-Lee, 2006). Notation3 was proposed by Berners-Lee as an alternative to the dominant notation for RDF which is RDF/XML. Notation 3 is a language which is more compact and probably better readable than RDF/XML, and is also extended to allow greater expressiveness.

UbiCore uses a basic 3-layer agent structure that is common to the APL approach (see e.g. Collier et al., 2005). The architecture can be seen in Figure 3. There is a behavior engine implemented in Java, a declarative middle-layer, and a set of so-called Reusable Atomic Behaviors (RABs). A RAB is a Java component and its main purpose is to act as sensor or actuator. However, we do not restrict RABs to be only sensors or actuators, i.e. components concerned with the agent's environment. A RAB can also be a reasoner (data-processor) if some of the logic needed is impossible or is not efficient to realize with S-APL, or if one wants to enable an agent to do some other kind of reasoning beyond the rule-based one.

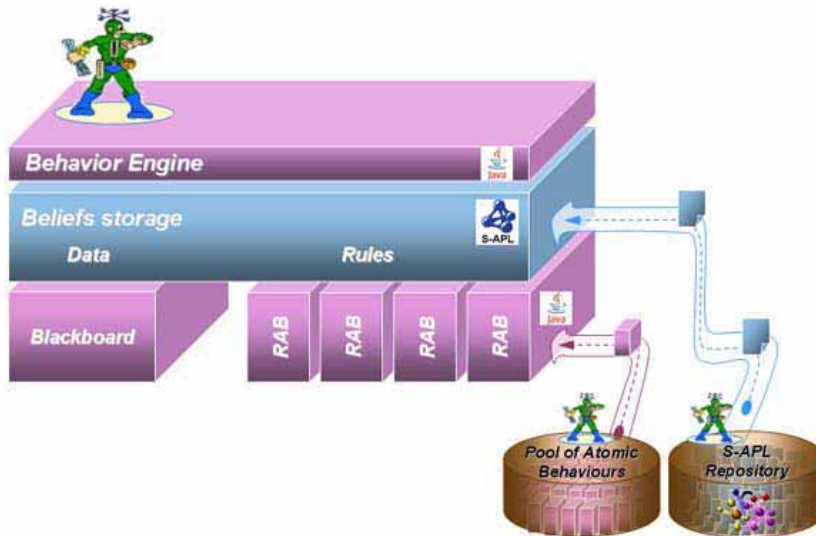


Fig. 3. The agent architecture.

The middle layer of the architecture acts as beliefs storage. What differentiates S-APL from traditional APLs is that S-APL is RDF-based. This provides the advantages of the semantic data model and reasoning. Another advantage of S-APL is that the data and the program code are treated in a similar way. Data and code use the same storage instead of two separate ones. This also means that: a rule upon its execution can add or remove another rule, the existence or absence of a rule can be used as a premise of another rule, and so on.

In traditional APL approaches this cannot be performed, because the rules are separated from the data. S-APL is very symmetric with respect to this – anything that can be done to a simple statement can also be done to any belief structure of any complexity. An S-APL agent can obtain data and rules also by querying S-APL repositories. For example a repository can contain a list of ready-made scripts that correspond to organizational roles. In our implementation, such querying is performed as inter-agent action with FIPA ACL messaging but does not involve any query or content languages beyond S-APL itself. As can be seen from Figure 3, agents also can load RABs remotely. This is done as an exception mechanism triggered when a rule prescribes engaging a RAB while the agent does not have it available. Thus, organizations are able to provide not only the rules to follow but also the tools needed for that.

An Ubiware-based application consists of a set of S-APL scripts (behavior models and data) and a set of Reusable Atomic Behaviors. In order to increase the reusability of the platform, we provide some ready-made scripts and behaviors. Therefore, our platform as such can be seen as consisting of the following three elements: (1) the behavior engine, (2) a set of “standard” SAPL models, (3) a set of “standard” RABs. The set of currently provided standard S-APL models includes for example communication models. The set of currently provided standard RABs includes RABs for such operations like downloading a web document, sending an email, loading and transforming into an S-APL presentation a text table document, loading an XML document, etc (see details in Katasonov, 2008).

Technically, our implementation is built on the top of the Java Agent Development Framework (JADE) (Bellifemine et al., 2007), which is a Java implementation of IEEE FIPA specifications. JADE provides communication infrastructure, agent lifecycle management, agent directory-based discovery and other standard services.

5. Intelligent resource visualization

5.1 Motivation

One of the reasons for intelligent resource visualization springs from the necessity of resource search and browsing processes enhancement. In (Marcos et al., 2005), several features of Classical Model of information search are criticized. First of all, this model does not adequately distinguish between the needs of a user and what a user must specify to get it. Very often, users may not know how to specify a good search query, even in natural language terms. Analyzing what is retrieved from the first attempt is usually not used to select useful results, but to find out what is there to be searched over. A second important criticism of the Classical Model is that any knowledge generated during the process of formulation a query is not used later in the sequence of search process steps, to influence the filtering step and presenting step of the search results, or to select the results. Finally, Classical Model provides an essentially context-free process. There is no proper way in which knowledge of the task context and situation, and user profile can have an influence on the information search process.

Thus, when we come to the vision of GUN (where all the resources of the virtual and the real world are connected and interoperate with each other) and the amount of information becomes huge, we have to elaborate new visualization techniques that simplify the search and browsing processes through reducing amount of queries via context-dependent resource visualization. Following this approach, we have a need to visualize the resource properties (in different from “directed arc between objects” representation way), the various relations between resources and the inter-resource communication process. And even more, we have a need to make this visualization more context-dependent, to be able to represent information in a handy way and adequate to a certain case (context), to reach the plasticity of UIs (Thevenin & Coutaz, 1999). Thus, the main focus in GUI development will be concentrated on the resource visualization aspects and we have a challenging task of semantically enhanced context-dependent multidimensional resource visualization.

5.2 4i: A new form of resource visualization

This part will study dynamic context-aware A2H (Agent-to-Human) interaction in UBIWARE, and will elaborate on a technology which we refer to as 4i (FOR EYE)

technology. 4i enables creation of a smart human interface through flexible collaboration of an Intelligent GUI Shell, various visualization modules, which we refer to as MetaProvider-services, and the resources of interest.

In UBIWARE, humans are important resources, which can play several distinct roles: a resource under care, a service provider, a user, and an administrator. Obviously, the humans need some graphical interfaces to interact with the rest of the system. The same person can play several roles, switch between them depending on the context, and, in result, require different interfaces at different times. In addition, a UBIWARE-based system presents a large integration environment with potentially huge amounts of heterogeneous data. Therefore, there is a need for tools facilitating information access and manipulation by humans. A semantic context-aware multimodal visualization approach would provide an opportunity for creating smart visual interfaces able of presenting relevant information in a more suitable and more personalized form.



Fig. 4. Human-Resource Adapter based on 4i (FOR EYE) technology.

From the GUN point of view, a human interface is a special case of a resource adapter (Human-Resource Adapter - HRAP). We believe, however, that it is unreasonable to embed all the data acquisition, filtering and visualization logic into such an adapter. Instead, external services and application should be effectively utilized. Therefore, the intelligence of a smart interface (HRAP) will be a result of collaboration of multiple agents: the human's agent, the agents representing resources of interest (those to be monitored or/and controlled), and the agents of various visualization services - MetaProviders via an agent of HRAP (see Figure 4). This approach makes human interfaces different from other resource adapters and indicates a need for devoted research.

Based on 4i technology, an infrastructure will be embedded into UBIWARE enabling effective realization of the following system functions:

- visualization of data provided by a service in response to a request;

- search, retrieving and visualization of data required by a human expert,
- providing access to contextual information, and visualization of it;
- visualization of resource registration, configuration, and security policy establishment processes;
- resource discovery via MetaProviders (because they act as thematic portals).

5.3 4i infrastructure

As was mentioned in the previous section, one of the requirements for visual interfaces is an ability to represent information regarding to chosen contextual property of a resource, an interface should allow a user to simply choose a context for data representation, and should even support cases of multiple contextual property selection for complex views and filtering purposes. Such requirements can be met by MetaProviders - sui generis portals of resources with specific visualization view. It is named MetaProvider in a sense that it provides an access and presents other resources, which in turn are providers of own information (data). All GUN resources have certain own location (physical and digital). But it does not mean that they should have just one way of getting access to them. MetaProvider is an interface-mediator that gives a possibility to mark/select a resource (object) on its interface and provide the link to original resource location. In other words, it allows resource registration for further access to its data. At the same time, any resource can be registered on variety of different MetaProviders in different views. The main feature of MetaProviders is that each party which takes care of some GUN-Resource registers the resource itself. It causes fast filling of information accessible through MetaProvider. And each user/resource in one moment has an access to related information of amount of others. But such interoperability brings a new requirement for the MetaProviders and users. They should share common ontology to be interoperable on semantic level. Additionally to semantic interoperability, GUN-Resources are proactive/goal-driven resources and supplied with a Resource Agent for resource-to-resource (R2R)/ agent-to-agent (A2A) communication.

4i (FOR EYE) is an ensemble of GUN Resource Platform Intelligent GUI Shell (smart middleware for context dependent use and combination of a variety of different MetaProviders depending on user needs) and MetaProviders, visualization modules/platforms that provide context-dependent filtered representation of resource data and integration on two levels (information/data integration of the resources to be visualized and integration of resource representation views with a handy resource browsing) (see Figure 5). Context-awareness and intelligence of such interface brings a new feature that gives a possibility to the user to get not just the raw data, but required integrated information based on a specified context. GUI Shell allows the user dynamic switching between MetaProviders for more suitable information representation depending on a context or resource nature. MetaProvider plays these four main roles:

- Context-aware resource visualization module that presents information regarding to specified context in more suitable and personalized for user form;
- Interface for integrated information visualization with intelligent context-aware filtering mechanism to present only relevant information, avoiding a glut of unnecessary information;
- Visual Resource Platform that allows resource registration, search, access and modification of needed information/data in a space of registered resources;
- Mediator that facilitates resource to resource (R2R) communication.

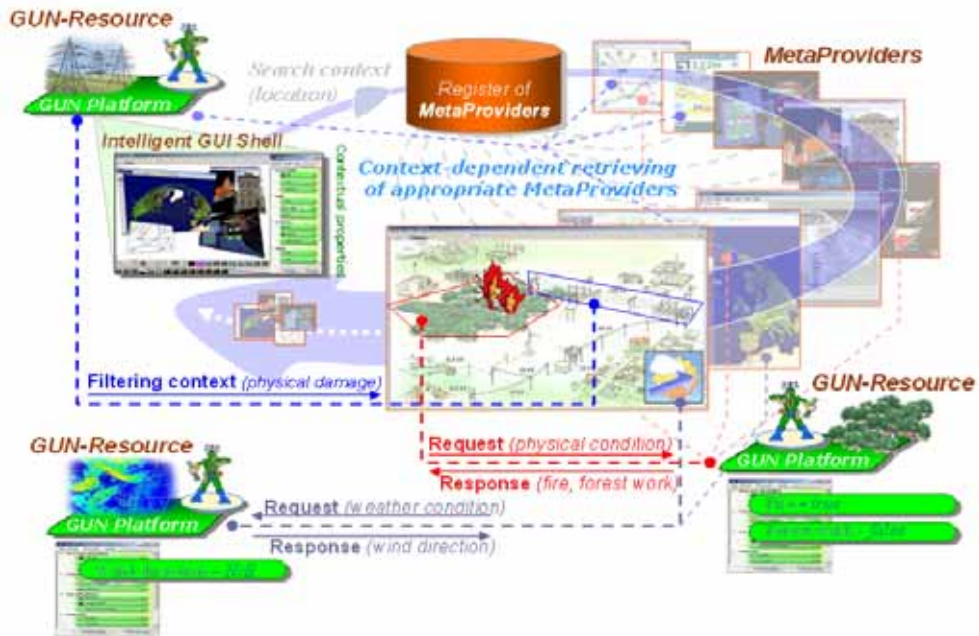


Fig. 5. Intelligent Interface for Integrated Information (4i technology).

5.4 Conclusion

4i (FOR EYE) technology is an integral part of UBIWARE. On one side, the technology is a base for Human-Resource adaptation and will be elaborated according to the principles and visions of UBIWARE. The intelligence of this smart interface is the result of collaboration of multiple agents: the human's agent, the agents representing resources of interest (those to be monitored or/and controlled or requesting a human), and the agents of various visualization services – MetaProviders via an agent of Human-Resource Adapter. From the other side, accordingly to the 4i, specific interfaces (MetaProviders) will be developed to provide functionality for a human (Platform administrator) to configure functionality of a UBIWARE-based system.

6. Resource histories

6.1 Motivation

In UBIWARE, every resource is represented by a software agent. Among major responsibilities of such an agent is monitoring the condition of the resource and the resource's interactions with other components of the system and humans. The beliefs storage of the agent will, therefore, naturally include the history of the resource, in a sense "blogged" by the agent. Obviously, the value of such a resource history is not limited to that particular resource. A resource may benefit from the information collected with respect to other resources of the same (or similar) type, e.g. in a situation which it faces for the first

time while others may have faced that situation before. Also, mining the data collected and integrated from many resources may result in discovery of some knowledge important at the level of the whole ubiquitous computing system. A scalable solution requires mechanisms for inter-agent information sharing and data mining on integrated information which would allow keeping the resource histories distributed without need to copy those histories to a central repository.

The work in this area is aimed at answering the following research question:

- How to semantically markup the history of a resource in a system, in order to make it reusable for other resources and at the system-level?
- What mechanisms are needed for effective and efficient sharing of information between the agents representing different resources?

6.2 General approach

The general approach, which was selected in this task, is to avoid designing some special protocols and languages for inter-agent information sharing, but rather to reuse as much as possible the tools developed as part of the UbiCore mentioned in the Section 4.

Obviously, there is a certain similarity in the following. On one hand, an agent always needs to query its own beliefs base in order to evaluate the left sides of its behavior rules in order to identify rules that are to be executed. On the other hand, when an agent asks another agent for some information, it, in a sense, queries the belief base of that other agent.

Our approach is therefore to design the external querying process so it would be almost the same as if the agent itself would query its belief base to check the conditions for executing a rule. This also means that we plan to use the Semantic Agent Programming Language (SAPL) not only as the means for prescribing the agents' behaviors, but also as the inter-agents communication content language (to be used instead of FIPA-SL or other languages of this type). The advantages of this should be obvious as the symmetry and expressive power in the UBIWARE platform will be maximized. The agents will be able to query each other not only for some facts (present or historical) about the external world (the domain) but also, for example:

- Query if the other agent knows a plan for achieving a certain goal,
- Query if the other agent knows a rule that should be applied in a particular situation.

6.3 Ontonuts

In this section we introduce Ontonuts concept to facilitate the presentation of modular scripts and plans in the UBIWARE platform. The idea is somewhat similar to the notion of Enterprise Java Beans, whereas it is semantic by nature and has its own architectural peculiarities. Instances of Ontonut concept in general represent a capability with known input and expected output. Then we adapt Ontonuts to the problem of distributed querying. There are two main viewpoints towards distributed querying in the UBIWARE: adapter-based and service-based. The former tackles the adaptation of data sources that are external to the platform (databases, files, web services, etc.), when the latter deals with the agent-to-agent querying. Nevertheless, both target the same goal: to make distributed querying transparent to an agent (see Figure 6).


```
O1 :type :Ontonut
O1 :precondition {A1 A2 A3}
O1 :effect {B1 B2 B3}
O1 :script {{A1 A2 A3}=>... =>{B1 B2 B3}}
```

```
O2 :type :Ontonut
O2 :precondition {B1 B2 B3}
O2 :effect {C1 C2 C3}
O2 :script {{B1 B2 B3}=>...=>{C1 C2 C3}}
```

The appearance of the goal G1 will activate Ontonuts engine that will match G1 against available effects and apply planning, which will result in an execution plan: O1=>O2=>G1.

6.4 Conclusion

The goal of this work is to structure the S-APL code of the UBIWARE agent in order to simplify programming of the agent and allow automated goal-driven planning. Although, the work applies to a narrow domain, it involves generic problems of agent planning and semantic programming. The main topic is the automated planning of distributed queries and inevitably interferes with distributed RDF queries and so-called virtual graphs, when the graph being queried does not have content beneath; instead, the RDF queries are used to generate SQL queries to the corresponding database.

The approach proposed here, does not try to map models, but uses patterns to define the desired result and applies queries or any other code to fill the patterns requested by the agent.

7. Middleware for Peer-to-Peer Discovery

7.1 Motivation

With the growing popularity of multi-agent systems, the number of deployed multi-agent platforms will continue to increase. We see the necessity of existence of mechanisms which allow creating connections between these platforms. We believe that UBIWARE should be extended with functionality which would provide agents issuing search request not only with local results, but also with global results from the entire network of interconnected platforms. Our goal is to find the way to extend the UBIWARE functionality so that it could handle inter-platform communication and discovery in peer-to-peer (P2P) manner. Such extension will allow us to create a connection between separate UBIWARE systems where agents residing on different platforms could exchange information and cooperate.

The goal is to design mechanisms which will extend the scale of semantic resource discovery in UBIWARE with peer-to-peer discovery. Such mechanisms have to enable an agent:

- To discover agents playing a certain organizational role,
- To discover an agent (or agents) possessing certain needed information,
- To discover resources (through its agents) of certain type or possessing certain properties (e.g. a device in state X, or a web-resource providing some information searched for).

In all cases, existence of a central Directory Facilitator is not assumed, so the discovery is to be performed in peer-to-peer manner. This has at least two goals:

- Improving the survivability of UBIWARE-based systems. P2P is a complementary mechanism which can be utilized in an exception situation where the central Directory Facilitator became for some reason unavailable.
- Discovery across UBIWARE-based systems. There could be several UBIWARE-based agent platforms (applications) started up independently, each having own Directory Facilitator. While JADE (underlying agent framework of UBIWARE) supports communication among agents residing on different platforms, it does not provide for the cross-platform discovery.

7.2 Inter-platform and agent discovery in multi-platform environment

Let us analyze the situation, where several independent UBIWARE-based agent platforms exist in the network, each having a set of agents running and registered with DF.

Within each platform, inter-agent discovery and communication is not a problem, since the platform's components take care of these acts. The question arises, would it be possible (and how) to allow the agents from different platforms to search remote DFs and interact with agents located on foreign platforms. Such solution deployed in UBIWARE platform would enhance its functionality – if agent cannot locate a service or obtain needed information within its home platform, it can query the DFs on known remote platforms in search of needed data. Agents would benefit from this possibility in two situations:

The service or information required by an agent cannot be located within the native platform,

An agent possessing such service/information no longer exists within the platform or cannot be contacted.

We see the similarities between peer-to-peer file sharing networks and multi agent systems. In P2P networks the nodes exchange messages in order to locate their neighbors and resources they share. In multi agent systems the agents attempt to discover other existing agents and services they provide in order to utilize them, thus enhancing the overall functionality of the platform. In this section we analyze the existing approaches used to build file sharing P2P networks that could be utilized to address the problem – the centralized approach of general Directory Facilitator and solutions known from pure peer-to-peer file sharing networks with connection to JADE's mechanism of DF federations.

The centralized approach for inter-platform agent discovery borrows from the hybrid model known from file sharing peer-to-peer networks implemented in Napster platform. In hybrid model the central server stores a list of available resources each node is sharing. Moving this idea to multi-platform environment, one of the existing platforms is designated to be a central server. Every agent wishing to publish its services contacts the central server and registers with the general Directory Facilitator. To discover an agent playing certain role, agents query the central server, which will return the list of addresses (AIDs) of agents providing required service. Figure 7 shows the idea of multi-platform environment with central Directory facilitator.

Fig. 7. Multi-platform environment with centralized DF.

The second way to achieve inter-platform agent discovery is through Federated Directory Facilitators. According to FIPA specifications, Directory Facilitators should be able to register with each other, creating federation within a single (also distributed) platform. When DF agent registers with another one, it becomes its child, and will from this point receive all search queries from its parent DF, also forwarding it to each of its child. The propagation of the query within federation is controlled by a parameter *MaxDepth* in search constraints, which specifies how many times the query should be forwarded. The parameter *MaxResults* limits the number of agents' descriptions that will be returned to agent issuing a query. Figure 8 shows an example of Agent Platform with multiple federated DFs each having set of agents registered with it.

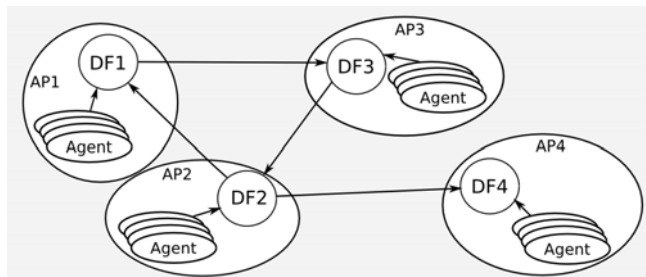


Fig. 8. Federation of Directory Facilitators.

The third way is to create a dynamic peer-to-peer topology. We borrow the idea of Gnutella system (Klingberg & Manfredi, 2002). Gnutella is a decentralized peer-to-peer system, which means it operates without presence of centralized server. Each participant acts as both server and a client; therefore the term "servent" was introduced. All servents share resources allowing others to locate and download them. By using messages passing system all servents execute two types of operation. First, by exchanging messages servents increase or maintain their degree of connectivity. Secondly, the search queries are exchanged in order

to locate required resources that may be shared by some servents connected to the network. The protocol requires that within the network exists at least one always-on node, which provides a new participant with addresses of the servents already operating. Each servent upon startup obtains a pool of addresses and connects to them. In order to discover other participants it starts the PING / PONG process, presented in Figure 9.

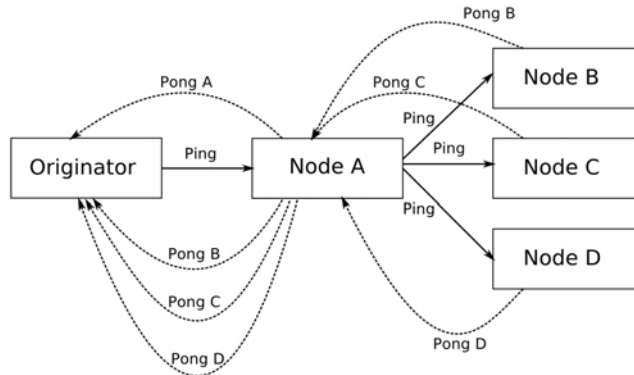


Fig. 9. Propagation of PING and PONG messages in Gnutella discovery process.

7.3 Conclusions and future work

We have presented three different approaches of building distributed peer-to-peer infrastructure in multiplatform environments. By the means of inter-platform discovery we give agents the opportunity to communicate, share services and resources beyond the boundaries of their home platforms.

Future work will include incorporating one of the described methods into the UBIWARE prototype. We also plan to conduct further research upon improving the efficiency of created network of agent platforms.

8. Conclusion and future work

In this chapter we present several challenges for achieving the vision of the Internet of Things and Ubiquitous Computing. Today's development in the field of networking, sensor and RFID technologies allows connecting various physical world objects to the IT infrastructure. However the complexity of such a system may become overwhelming and unmanageable. Therefore there is a need for computing systems capable of "running themselves" with minimal human management which is mainly limited to definition of some higher-level policies rather than direct administration. We believe that this complexity can be solved by incorporating the principles of multi-agent systems because of its ability to facilitate the design of complex systems.

Another challenge that has to be faced is the problem of heterogeneity of resources. Semantic technologies are viewed today as a key technology to resolve the problems of interoperability and integration within heterogeneous world of ubiquitously interconnected objects and systems. Semantic technologies are claimed to be a qualitatively stronger approach to interoperability than contemporary standards-based approaches. For this

reason we believe that Semantic Web technologies will play an important role in the vision of Internet of Things.

We do not believe that imposing some rigid standards is the right way to achieve the interoperability. Instead of that we suggest using middleware that will act as glue joining heterogeneous components together.

Based on these beliefs we describe our vision of such a middleware for the Internet of Things, which has also formed the basis for our research project Ubiware. Ubiware is one of the steps needed to achieve a bigger vision that we refer to as Global Understanding Environment (GUN). Global Understanding Environment (GUN) aims at making heterogeneous resources (physical, digital, and humans) web-accessible, proactive and cooperative. Three fundamentals of such platform are Interoperability, Automation and Integration.

The most important part of the middleware is the core. In the Ubiware project we refer to it as UbiCore. The goal of UbiCore is to give every resource a possibility to be smart (by connecting a software agent to it), in a sense that it would be able to proactively sense, monitor and control its own state, communicate with other components, compose and utilize own and external experiences and functionality for self-diagnostics and self-maintenance.

In order to be able to describe our intentions we needed a language. There are several existing agent programming languages (APLs) like AGENT-0, AgentSpeak(L), 3APL or ALPHA. All of those are declarative rule-based languages and are based on the first-order logic of n-ary predicates. All of them are also inspired by the Beliefs-Desires-Intentions architecture. However none of them considers the possibility of sharing the APL code with other agents or leaving the agent in the run-time.

Export and sharing of APL code would, however, make sense because of two main reasons. Firstly, this approach can be used for specifying the organizational roles since organizational roles are specified with a set of rules and APL is a rule-based language. Secondly, the agents may access a role's APL code not only in order to enact that role, but also in order to coordinate with the agents playing that role. In this way an agent can communicate its intentions with respect to future activities.

When thinking about using the existing APLs in way that mentioned above, there are at least two issues present. Firstly, the code in an APL is, roughly speaking, a text. However in complex systems, a description of a role may need to include a huge number of rules and also a great number of beliefs representing the knowledge needed for playing the role. Therefore, a more efficient, e.g. a database-centric, solution is probably required. Secondly, when APL code is provided by an organization to an agent, or shared between agents, mutual understanding of the meaning of the code is obviously required.

As a solution to these two issues, we see creating an APL based on the W3C's Resource Description Framework (RDF). RDF uses binary predicates only, i.e. triples. Our proposition for such an RDF-based APL is the Semantic Agent Programming Language (S-APL). We decided to use Notation3 as the base of this language because it is compact and better readable than RDF/XML.

We use a basic 3-layer agent structure that is common for the APL approach. There is a behavior engine implemented in Java, a declarative middle-layer, and a set of sensors and actuators which are again Java components. The latter we refer to as Reusable Atomic Behaviors (RABs). In general a RAB can be any component concerned with the agent's

environment, i.e. reasoner. The middle layer is the beliefs storage. What differentiates S-APL from traditional APLs is that S-APL is RDF-based. This provides the advantages of the semantic data model and reasoning.

The architecture of our platform implies that a particular application utilizing it will consist of a set of S-APL documents (data and behavior models) and a set of atomic behaviors needed for this particular application. There is a set of standard RABs and a set of standard S-APL scripts. They create the base of the Ubiware core. On top of them, the user can specify his/her own S-APL scripts and/or RABs.

We believe that the vision of Internet of Things also needs a new approach in the field of resource visualization. The classical model of information search has several disadvantages. Firstly, it is difficult for the user to transform the idea of the search into the proper search string. Many times, the first search is used just to find out what is there to be searched. Secondly, the classical model introduces a context-free process.

In order to overcome these two disadvantages of the classical model, we introduce For Eye (4i) concept. 4i is studying a dynamic context-aware A2H (Agent-to-Human) interaction in Ubiware. 4i enables the creation of a smart human interface through flexible collaboration of an Intelligent GUI Shell, various visualization modules, which we refer to as MetaProvider-services, and the resources of interest.

MetaProviders are visualization modules that provide context-dependent filtered representation of resource data and integration on two levels - data integration of the resources to be visualized and integration of resource representation views with a handy resource browsing. GUI Shell is used for binding MetaProviders together.

The fact that all resources are represented by an agent responsible for this resource implies that such an agent has knowledge of the state of this resource. The information about this state may be beneficial for other agents. Other agents can use this information in a situation which they face for the first time while others may have faced that situation before. Also, mining the data collected and integrated from many resources may result in discovery of some knowledge important at the level of the whole ubiquitous computing system.

We believe that the creation of a central repository is not the right approach. Instead of that we propose the idea of distributed resource histories based on a transparent mechanism of inter-agent information sharing and data mining. In order to achieve this goal we introduce the concept of Ontonut.

The Ontonuts technology is implemented as a combination of a Semantic Agent Programming Language (S-APL) script and Reusable Atomic Behaviors (RABs), and hence, can be dynamically added, removed or configured. Each Ontonut represents a capability of accessing some information. An Ontonut is annotated by precondition, effect and script property. Precondition defines a state required for executing the functionality of desired ontonut. Effect defines the resulting data that can be obtained by executing this Ontonut. The script property defines the way how to obtain the data. A part of the Ontonuts technology is also a planner that automatically composes a querying plan from available ontonuts and a desired goal specified by the agent.

In the future several Ubiware-based platforms may exist. Our goal is to design mechanisms which will extend the scale of semantic resource discovery in Ubiware with peer-to-peer discovery. We analyzed three approaches: Centralized Directory Facilitator, Federated Directory Facilitators and creation of a dynamic peer-to-peer topology. We believe that this

type of discovery should not be based on a central Directory Facilitator. This will improve the survivability of the system.

In the future we would like to concentrate on the core extension. Currently we are working on an extension for agent observable environment. This opens new possibilities for coordination and self-configuration. In the area of peer-to-peer inter-platform discovery we plan to conduct further research on improving the efficiency of created network of agent platforms. Another topic that we are researching is the area of self-configuration and automated application composition.

9. References

- Bellifemine, F. L., Caire G., Greenwood, D. (2007). *Developing Multi-Agent Systems with JADE*, Wiley, ISBN 978-0470057476
- Berners-Lee, T., (2006). *Notation 3*, online (May 2009): <http://www.w3.org/DesignIssues/Notation3.html>
- Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., and Mylopoulos, J. (2004) Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems* 8(3): 203-236
- Brock, D.L., Schuster, E. W., Allen, S.J., and Kar, Pinaki (2005) An Introduction to Semantic Modeling for Logistical Systems, *Journal of Business Logistics*, Vol.26, No.2, pp. 97-117 (available in: <http://mitdatacenter.org/BrockSchusterAllenKar.pdf>).
- Buckley, J. (2006) From RFID to the Internet of Things: Pervasive Networked Systems, *Final Report on the Conference organized by DG Information Society and Media, Networks and Communication Technologies Directorate, CCAB, Brussels* (online :http://www.rfidconsultation.eu/docs/ficheiros/WS_1_Final_report_27_Mar.pdf).
- Collier, R., Ross, R., O'Hare, G. (2005). Realising reusable agent behaviours with ALPHA. In: Eymann, T., Klugl, F., Lamersdorf, W., Klusch, M., Huhns, M.N. (eds.) *MATES 2005*. LNCS (LNAI), vol. 3550, pp. 210-215. Springer, Heidelberg
- Dastani, M., van Riemsdijk, B., Dignum, F., Meyer, J.J. (2004). A programming language for cognitive agents: Goal directed 3APL. In: Dastani, M., Dix, J., El Fallah-Seghrouchni, A. (eds.) *PROMAS 2003*. LNCS (LNAI), vol. 3067, pp. 111-130. Springer, Heidelberg
- Jennings, N.R., Sycara K. P., and Wooldridge, M. (1998). A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems* 1(1): 7-38.
- Jennings, N.R. (2000) On agent-based software engineering. *Artificial Intelligence* 117(2): 277-296.
- Jennings, N.R. (2001) An agent-based approach for building complex software systems. *Communications of the ACM* 44(4): 35-41.
- Katasonov, A. (2008). *UBIWARE Platform and Semantic Agent Programming Language (S-APL). Developer's guide*, Online: <http://users.jyu.fi/~akataso/SAPLguide.pdf>.
- Kaykova O., Khriyenko O., Kovtun D., Naumenko A., Terziyan V., and Zharko A. (2005a) General Adaption Framework: Enabling Interoperability for Industrial Web Resources, In: *International Journal on Semantic Web and Information Systems, Idea Group*, Vol. 1, No. 3, pp.31-63.
- Kephart J. O. and Chess D. M. (2003). The vision of autonomic computing, *IEEE Computer*, Vol. 36, No. 1, pp. 41-50

- Klingberg, T., Manfredi, R. (2002) *Gnutella 0.6*, online: http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html
- Langegger, A., Blochl, M., Woss, W., (2007). Sharing Data on the Grid using Ontologies and distributed SPARQL Queries, *Proceedings of 18th International Conference on Database and Expert Systems Applications*, pp.450-454, Regensburg, Germany, 3-7 Sept. 2007
- Lassila, O. (2005a) Applying Semantic Web in Mobile and Ubiquitous Computing: Will Policy-Awareness Help?, in Lalana Kagal, Tim Finin, and James Hendler (eds.): *Proceedings of the Semantic Web Policy Workshop, 4th International Semantic Web Conference*, Galway, Ireland, pp. 6-11.
- Lassila, O. (2005b) Using the Semantic Web in Mobile and Ubiquitous Computing, in: Max Bramer and Vagan Terziyan (eds.): *Proceedings of the 1st IFIP WG12.5 Working Conference on Industrial Applications of Semantic Web*, Springer IFIP, pp. 19-25.
- Lassila, O., and Adler, M. (2003) Semantic Gadgets: Ubiquitous Computing Meets the Semantic Web, In: D. Fensel et al. (eds.), *Spinning the Semantic Web*, MIT Press, pp. 363-376.
- Mamei, M, Zambonelli F. (2006). *Field-Based Coordination for Pervasive Multiagent Systems*, Soringer, ISBN 9783540279686, Berlin
- Quilitz, B., Leser, U. (2008) Querying Distributed RDF Data Sources with SPARQL, *The Semantic Web: Research and Applications, 5th European Semantic Web Conference, ESWC 2008*, Tenerife, Canary Islands, Spain, June 1-5, 2008, pp.524-538.
- Rao, A.S. and Georgeff, M.P.(1991) Modeling rational agents within a BDI architecture. *Proc. 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pp. 473-484.
- Rao, A.S. (1996) AgentSpeak(L): BDI agents speak out in a logical computable language. *Proc. 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, LNCS vol.1038, pp. 42-55.
- Tamma, V.A.M., Aart, C., Moyaux, T., Paurobally, S., Lithgow-Smith, B., and Wooldridge, M. (2005) An ontological framework for dynamic coordination. *Proc. 4th International Semantic Web Conference'05*, LNCS vol. 3729, pp. 638-652.
- Terziyan V. (2003) Semantic Web Services for Smart Devices in a "Global Understanding Environment", In: R. Meersman and Z. Tari (eds.), *On the Move to Meaningful Internet Systems 2003: OTM 2003 Workshops*, Lecture Notes in Computer Science, Vol. 2889, Springer-Verlag, pp.279-291.
- Terziyan V. (2005) Semantic Web Services for Smart Devices Based on Mobile Agents, In: *International Journal of Intelligent Information Technologies*, Vol. 1, No. 2, Idea Group, pp. 43-55.
- Thevenin, D. and Coutaz, J., (1999). Plasticity of User Interfaces: Framework and Research Agenda. In *Proceedings of Interact'99*, vol. 1, Edinburgh: IFIP, IOS Press, 1999, pp. 110-117.
- Vázquez-Salceda, J., Dignum, V., and Dignum, F. (2005) Organizing multiagent systems. *Autonomous Agents and Multi-Agent Systems* 11(3): 307-360.
- Wooldridge, M. (1997) Agent-based software engineering. *IEE Proceedings of Software Engineering* 144(1): 26-37.