CONTEXT-POLICY-CONFIGURATION
Paradigm of Intelligent Autonomous System Creation

Oleksiy Khriyenko, Sergiy Nikitin
Industrial Ontologies Group, Agora Center, University of Jyväskylä, P.O. Box 35(Agora), FIN-40014 Jyväskylä, Finland

Vagan Terziyan
Industrial Ontologies Group, MIT Department, University of Jyväskylä, P.O. Box 35(Agora), FIN-40014 Jyväskylä, Finland

Keywords: Policy, Policy-based Configuration, Autonomous System Collaboration, Context Awareness of System Behaviour.

Abstract: Next generation of integration systems will utilize different methods and techniques to achieve the vision of ubiquitous knowledge: Semantic Web and Web Services, Agent Technologies and Mobility. Nowadays, unlimited interoperability and collaboration are the important things for industry, business, education and research, health and wellness, and other areas of people life. All the parties in a collaboration process have to share data as well as information about actions they are performing. During the last couple of years, policies have gained attention both in research and industry. Policies are considered as an appropriate means for controlling the behaviour of complex systems and are used in different domains for automating system administration tasks (configuration, security, or Quality of Service (QoS) monitoring and assurance). The paper presents Semantic Web driven approach for context-aware policy-based system configuration. Proposed Context-Policy-Configuration approach for creation of intelligent autonomous systems allows system behaviour modification without source code change or requiring information about the dependencies of the components being governed. The system can continuously be adjusted to externally imposed constraints by policy determination and change.

1 INTRODUCTION

During the last tens year's humankind utilizes computers and numerous research results in the area of information technologies to build intelligent systems that help people in various domains: professional activities, entertainments, social sphere and etc. Experts build such systems in different domains to solve various tasks. But, all of them use models of data and knowledge representation as a basis for system creation.

With an intensive development of the Internet and very fast growing amount of information and data world wide, semantic technologies have become very popular. To achieve the vision of ubiquitous knowledge, the next generation of integration systems will utilize different methods and techniques. These include Semantic Web (Semantic Web, 2001, Berners-Lee et al., 2001) and Web Services, Agent Technologies, Mobility (Curbera et al., 2002, Clabby, 2002), and WebServices (Ankolekar et al., 2002, Paolucci et. al., 2002, FIPA, 2001). Semantic technologies are viewed today as a key technology to resolve the problems of interoperability and integration within the heterogeneous world of ubiquitously interconnected objects and systems. But still, aspects such as context and proactivity of these resources and systems are quite in demand nowadays and should be considered more comprehensively. Semantic technologies are a qualitatively stronger approach to interoperability than contemporary standards based approaches.

At the same time, to make system really intelligent, dynamic and autonomous, we have to utilize Agent Technologies. The vision of autonomic computing emphasizes that the run-time self-manageability of a complex system requires its components to be, to a certain degree autonomous themselves. Software agent technologies will play an important part in building such complex systems. Agent based approach to software engineering is also considered to be facilitating the design of complex systems. When it comes to developing complex, distributed software based systems, the agent based approach is advocated (Jennings, 2001). From the implementation point of view, agents are the 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).

To be smart, system should be able to take into account current state of environment, react on changes of it and behave accordingly. Thus, there is a need to elaborate such a framework of system development, which allow us to build an influence model of certain situation not only on single entity, but also on a system as whole.

From other side, to provide an ability to the system to be automatically controllable, policy based approach may be highly valuable. Policies are “rules governing choices in the behaviour of a system” (Damianou et. al., 2001). They contain the logic for guiding decisions during the execution of the system. Through policy, people can precisely express bounds on autonomous behaviour (permissions) and expectations of performance (obligations) in a way that is consistent with their appraisal of an agent’s competence in a given context. Policies allow changing the behaviour of a system without changing low-level code, creating more adaptable systems whose behaviour can be altered dynamically. The ability to change policies dynamically means that poorly performing agents can be immediately brought into compliance with corrective measures.

Policies can be used in different areas and domains. For example, policies have been used for access control, network and quality-of-service management, user preferences, operational policies, storage management, system configuration, self-management, multi-agent systems, etc. The policy community has been researching topics like policy specification, management, enforcement, reasoning, negotiation, refinement, discovery and many others. Policies can be expressed at different levels, referred to as a policy hierarchy, ranging from high-level abstract policies over specification-level policies to low-level configuration policies.

Among other benefits of policy-based approaches, there are: reusability, efficiency, extensibility, context-sensitivity, verifiability, support for both simple and sophisticated components, and external reasoning about component behavior. Policies can be used to explicitly express agreements, conventions, precedents, and salience conditions that help make automated components more effective players. They can be used to enforce bounds and expectations that increase interpredictability, they can be used to establish and maintain common ground, and their ability to be imposed and adjusted at runtime enables dynamic directability.

2 CONTEXT-POLICY CONFIGURATION PARADIGM

Developing and maintaining large-scale, distributed applications is a complex task. Middleware has traditionally been used to simplify application development by hiding low-level details and by offering generic services that can be reused and configured by application developers. However, middleware technology has not kept up with the growing demands that emerge in the digital society: the scale of distributed applications is rapidly increasing, the range of users that compose and configure applications has expanded significantly, the increased scope of distributed applications has also resulted in more advanced application composition scenarios.

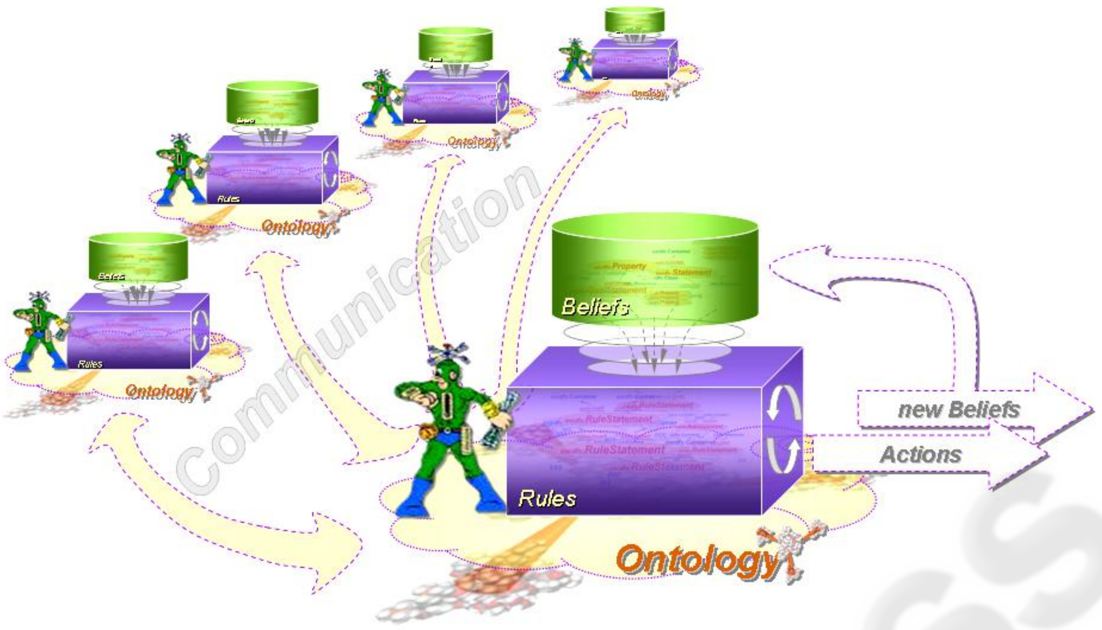
We are basing our research on UBIWARE Platform. The UBIWARE Platform is a development framework for creating multi-agent systems. It is built on the top of the Java Agent Development Framework JADE[1], which is a Java implementation of IEEE FIPA specifications. The name of the platform comes from the name of the research project, in which it was developed. In the UBIWARE project[2], a multi-agent system was seen, first of all, as a middleware providing interoperability of heterogeneous (industrial) resources and making them proactive and in a way smart.

2.1 Proactive Goal-driven Resource as a Main Entity of Any System…

A resource of a new Web is a *proactive* goal-driven *dynamic* entity that adequately and proactively

[1] JADE - http://jade.tilab.com/

[2] UBIWARE project - http://www.cs.jyu.fi/ai/OntoGroup/UBIWARE_details.htm



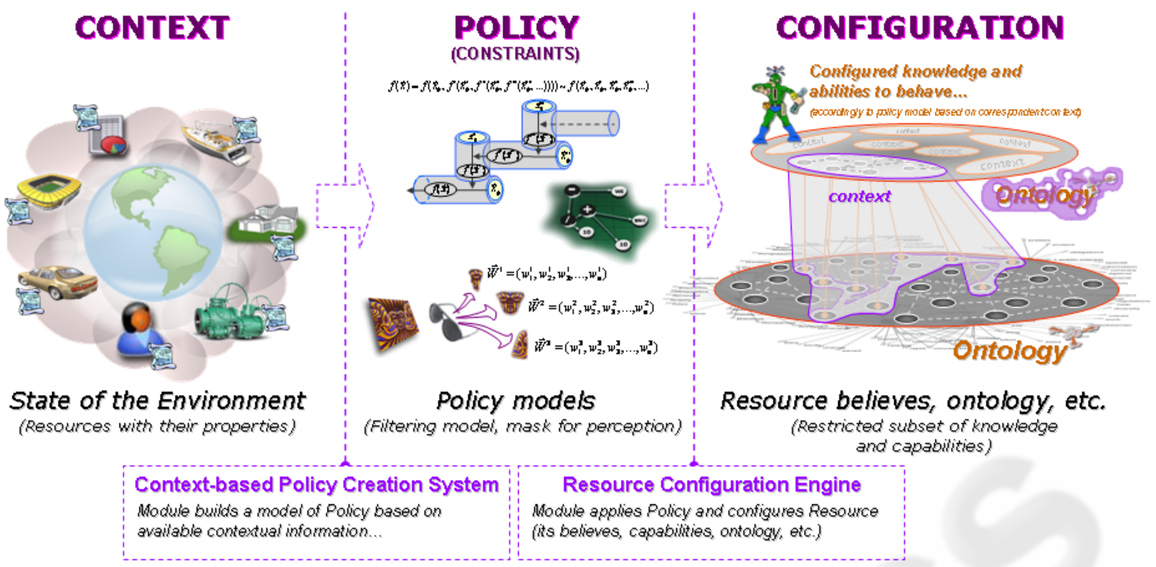
Picture 1: Agent environment structure.

reacts on changes within its external environment or within itself. As a consequence of resource dynamism and proactivnes, environment itself becomes more dynamic. Following GUN (Kaykova et. al., 2005) vision (Global Understanding Environment, where all the resources of the virtual and the real world are connected and interoperate with each other) that considers “everything as a resource” (even abstract entities) with correspondent semantic annotation and adapter, accompanied with an agent to be proactive, dynamic and autonomous.

Accordingly to semantic extension of agent programming language Semantic Agent Programming Language (S-APL) (Katasonov and Terziyan, 2007) that solves such issues as description of rules and beliefs (representing the knowledge needed for playing the role) and understanding of the semantics of the rules, the meaning of predicates used in those rules by all the parties involved while using first-order logic as the basis for an APL, we are considering three agent language constructs: Belief, Goal and Behavior.

Figure (Figure 1) shows us a structure of an agent environment. Belief Storage is represented by statements about environment, resource and agent (as well as about other resources and agents). All this information can be considered as preconditions and input data for rules (behaviours) execution. Then Rule Engine all the time is checking availability of the rules that can be performed and result changes within the beliefs as well as actions with correspondent beliefs changes. Resource’s behaviour results to changes in the Resource itself and to changes in the environment. In return, changes in the environment influence on the Resources. Thus, such dynamism and proactivnes bring context-awareness to the system, and more and more statements and behaviours become context-dependent. Thus, in this architecture the trigger that runs agent behaviour is a goal that should be specified for the agent. Based on specified goal, a behaviour planning process builds the behaviour model that is presented by appropriate set of behavioural rules that leads agent towards the set goal. Later such an abstract plan should be bound with particular instances from the agent beliefs or those that can be obtained via inference in runtime. Such rule based behavior representation present us rule based programming approach where conditional part of the rule is written in a way of semantic description that brings much more flexibility comparing to hardcoded programming.

Applying this approach we are able to build a system with two different levels of programming/administration. First level is the level of “advanced user” programming/administration and implies building of the rules to reach different goals that cover particular domain. It means that we define certain domain by Ontology of Goals - set of possible abstract goals that can be reached by


CONTEXT
POLICY
(CONSTRAINTS)
CONFIGURATION
Configured knowledge and abilities to behave...
context
Ontology
Ontology
State of the Environment
(Resources with their properties)
Policy models
(Filtering model, mask for perception)
Resource believes, ontology, etc.
(Restricted subset of knowledge and capabilities)
Context-based Policy Creation System
Module builds a model of Policy based on available contextual information...
Resource Configuration Engine
Module applies Policy and configures Resource (its believes, capabilities, ontology, etc.)

Picture 2: Context-Policy-configuration Paradigm.

resource (including sub-goal hierarchy), and by set of abstract Behaviour Rules that can be used to achieve these goals (sub-goals) (Khriyenko and Terziyan, 2006).Here we have abstraction in sense that Goals and Rules defined by variables without specifying concrete particular instances. Further these Rules can be used by planning module to build behavior plan to reach a goal if some abstract plans for goal achievement are not predefined beforehand. After completing this stage, our Behaviour/Rule Engine can build behavior plans and act accordingly to achieve any abstract goal. Thus, we come to the high-level system programming/administration stage where user has to put the constraints on abstractions, he/she should specify/create concrete instance of goal/goals and provide necessary initial states of the system. During initialization of variables in such abstraction user may specify concrete instance or set a range of possible values. In words of Semantic Web approach, user defines domains and ranges that can be considered as a policy (*D&R-type* policy) for the system that should be followed during goal achievement process. Those entities, which are not a matter of principle of user, can be left unspecified and will be found and bound by Engine automatically as a semantically close entities based on their semantic description (due to this approach, we have such opportunity). But, even in this case, user should specify levels of significance for entity's properties that will be taken into account by Engine. Thus, there is another type of policy, which is a kind of vector of weights of properties' significance (in this case) or vector of weights of any abstraction generally ($\vec{W}$ *-type* policy).

The main duty of "high-level user" is to define constraints/policies for a system to solve particular tasks among huge amount of possible task that can be performed by system. In this case, user can be considered as a certain particular context to the system. Generally, any belief (state of the environment or any Resource, with their properties and etc.) can be considered as contextual information to the system and put constraints/policies on actions and resources that are used during a goal achievement process. This implies restriction of Ontology (used by System): classes, properties, their domain and ranges. Restriction of classes and properties means either full prohibition of use or setting a level of preference for correspondent entity. For such a restriction, we can use $\vec{W}$ *-type* policy with the values of weights from the interval [0..1]. Depending on a context, different properties become more or less relevant that gives us different vectors of weights. To make a system autonomous and able to configure itself depending on contextual information, we have to supply the system with a pool of rules that will take contextual information as a conditional part and apply correspondent policies to system behaviour and used ontology (see Figure 2). Thus, Context-dependent Policy-based Control is a very promising approach, able to leave Resource flexible, dynamic and controlled at the same time. With this approach we do not program system in a hardcoded way, but build it able to change internal functionality and behaviour on the fly when context is changed.

2.2 Role-based Policy Control of a System

Generally we deal with a system with big amount of entities (Resources) with own behaviours and goals. To be able to control the system on general level, we have to put constraints/policies on separate entities as well as on the system in whole. System that is based on full unlimited knowledge (Ontology and Rules) and has an ability to make any actions is a Global (Mother) System. This system is unrestricted by any specific goal and able to behave towards achieving all possible goals. Any organization, union, company, society, group, individual and etc. can be considered as a sub system that plays certain Role, which restricts it with particular set of goals and knowledge/resources used for goal achievement (see Figure 3). Here Role of a system is a context that configures it via applying correspondent policy. Thus, any context is a kind of Role that implies (re-)configuration of a system.

Creation of Roles is a duty of “advanced user/administrator” that creates correspondent set of policies and rules of their application. In case, Role has been applied and there is still a need to make goal definition more concrete, then “high-level user/administrator” can be asked to do this.

Coming back to the approach of autonomous goal-driven Resource, application of new Role to the Resource Agent means creation of new working area for it, configured accordingly to correspondent Policy that restricts its’ degree of freedom for planning and execution processes. In this case Resource can play several Roles and share available resources based on own configured knowledge that corresponds to the Role-related area. Concerning the intelligent part of the Resource, the main aspect is decision making. It is an ability to decide which one among possible behaviours/actions is the best one in particular situation, is profit estimation of one or another plan among set of possible to be performed. To make a decision we have to weight different actions and deduct which of them are more profitable/advantageous for the Resource depending on its’ goal. Such technique is widely used in Game Theory[3] that attempts to mathematically capture behavior in strategic situations, in which an individual's success in making choices depends on the choices of others. There is a possibility to build different models to rank behaviours (from simple to very complex calculation methods). These methods should take into account context of current situation to simplify calculation process and provide us more relevant results. In another words, methods should follow applied W *-type* policy (vector of priorities of possible behaviours) defined by Role. It may happen that Roles become contradicted in sense of contradicted policies. In such situation Agent should refuse one or several contradicted Roles to avoid any contradictions in the system, or play all Roles if contradictions are not strong and concern only priorities of behaviours and goals. In the last case we have a deal with nested Policy-based control and new type of policy that regulates priority between lower level Policies.

2.3 Policy Models in Use

Now, when we highlighted several areas and tasks that need context-dependent policy-based control, let us come to elaboration of a general vision of Context-Policy-Configuration paradigm and show models in use.

In the last decade, several policy description languages have been developed, mostly designed for specific purposes, including:

- Ponder (Damianou et. al., 2001) is a policy language for specifying authorization and obligation policies in the context of distributed networks and systems;
- The eXtensible Access Control Markup Language (XACML) (OASIS, 2005) standard includes an XML-based policy language for specifying authorization policies;
- KAoS (Uszok et. al., 2003) and Rei (Kagal et. al., 2003) are semantic policy languages for expressing authorization and obligation policies. They both allow the inclusion of external ontologies, defining the semantics of domain-specific concepts;
- The Web Service Policy framework (WS-Policy) (IBM et. al., 2006) provides an extensible grammar for expressing non-functional requirements for interacting with a web service;
- GlueQoS (Wohlstadter et. al., 2004) is an extension of WS-Policy for specifying quality-of-service (QoS) features of web services.

Expectations regarding the new generation of Web depend on the success of Semantic Web technology. Resource Description Framework (RDF) is a basis for explicit and machine-readable representation of semantics.

[3] Game Theory
http://en.wikipedia.org/wiki/Game_Theory

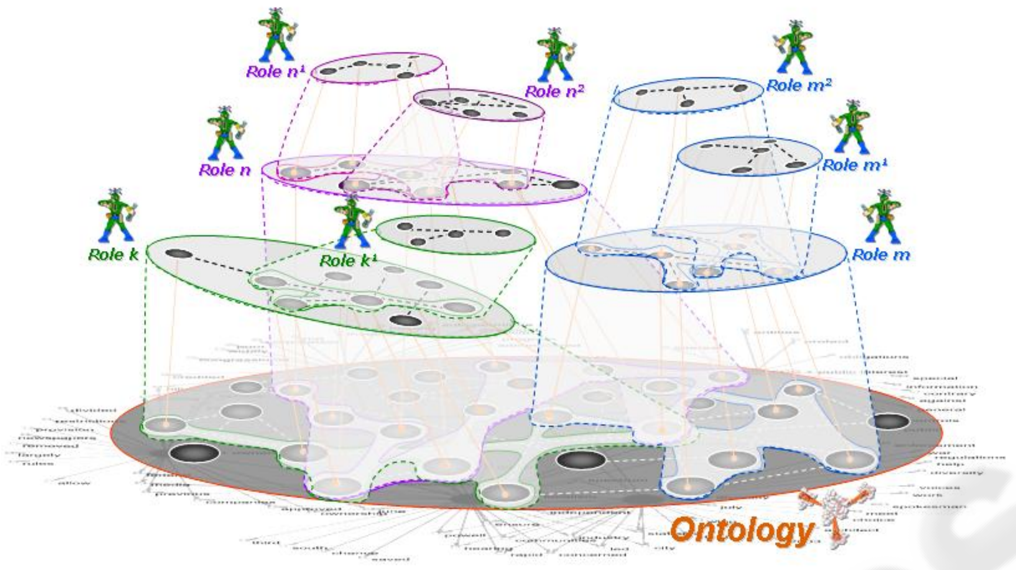


Figure 3: Role-based System creation.

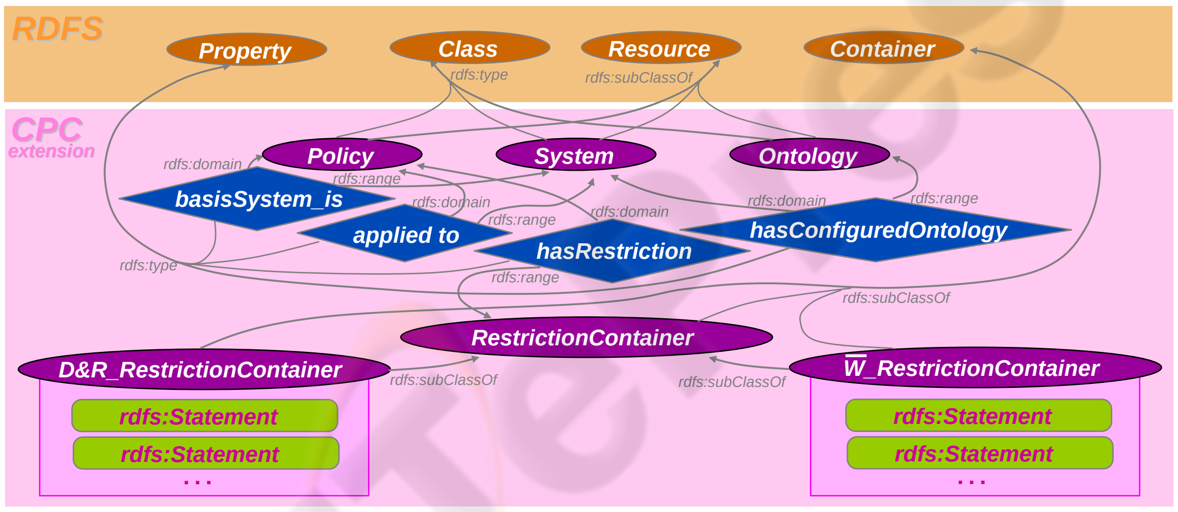


Figure 4: Initial part of Context-Policy-Configuration extension of RDFS.

To be compatible with widely used technology we extend RDF Schema with some classes and properties for policy description. Figure 4 shows us initial part of CPC-extension of RDFS. In the platform we utilize N3 representation in S-APL language.

Let us follow an example to show main principles of policy based system configuration. For example (see Figure 5) consider a System – “GreenFactory” as a subsystem of System - “Factory” with only difference that “GreenFactory” utilized only green kind of energy: Nuclear-, Hydro-, Wind-, Sun-energy, etc. Here we can use a D&R-type policy that restricts the ontology of mother-system (“Factory”) and redefines a range of the “useEnergy” property for the “GreenFactory”.

Figure shows us the range of the “useEnergy” property in “Ontology #i” that is used by system - “Factory #n”. The range is presented by list of different kinds of energies (Oil-, Wood-, Coal-, Nuclear-, Hydro-, Wind-, Sun-energy). Statement #1 states that “Policy #n” is applied to the system “GreenFactory #m” in case if this system plays the role “Role #k”.

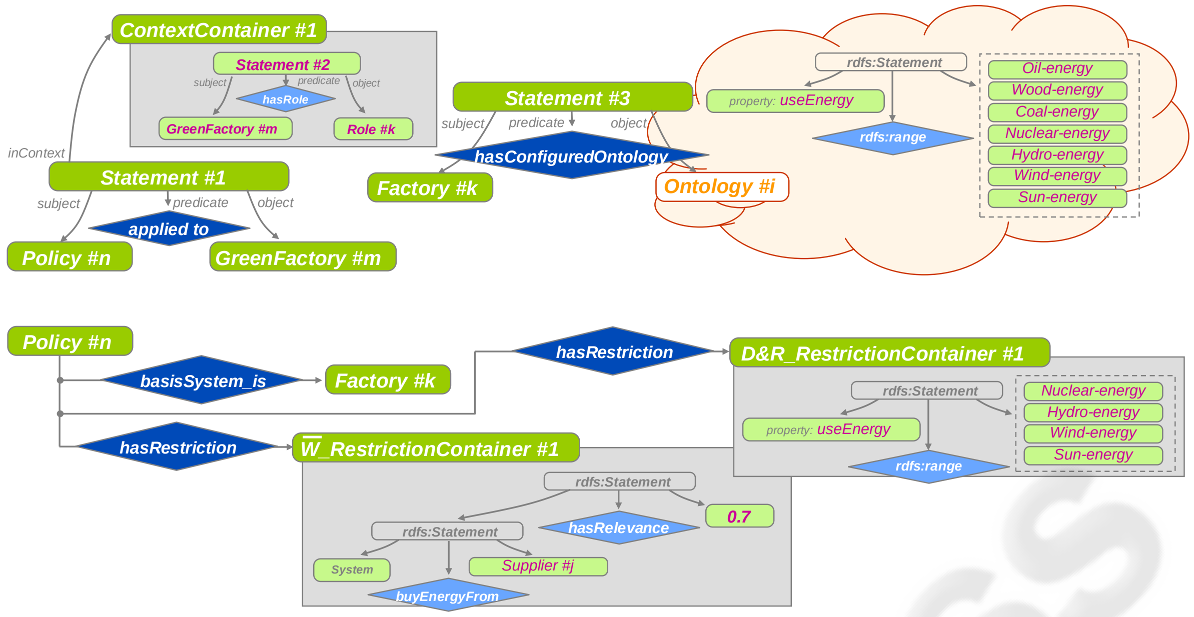


Figure 5: Policy model in use.

In other words, fact that system “GreenFactory #m” should play the role “Role #k” is a context/condition for the system agent to apply the policy “Policy #n” and to configure the system accordingly to the policy. From the policy description statements we can see that “Factory #k” system is a basis for policy application. As one of the restrictions considered by policy “D&R_RestrictionContainer #1” contains statement that redefines range for the “useEnergy” property and limits it to Nuclear-, Hydro-, Wind- and Sun-energies.

Now let us consider a case when “GreenFactory” join some industrial financial group and should follow a policy that demands at least 70% of energy to be bought from the energy supplier that belongs to the same financial group even if it is more expensive then buy energy from other suppliers. In this case $\vec{W}$ -type policy will be applied to the “GreenFactory” system and relevance of the statement which states that “GreenFactory” buys energy from correspondent supplier is 0.7 and relevance of others is 0.3. From the Figure 5 we see that statement in “$\vec{W}$ _RestrictionContainer #1” defines that statement, which states that system buys energy from supplier “Supplier #j”, has relevance equal 0.7. Relevance of the statements with other suppliers can be allocated among of them equally or according to other policies and goals of the system.

Thus, after policy applying, system agent configures system ontology and own believes to behave accordingly to correspondent context and applied policy.

3 CONCLUSIONS

Although the flexibility of agent interactions has many advantages when it comes to engineering a complex system, the downside is that it leads to certain unpredictability of the run-time system. Emergence of a solution that would allow flexible yet predictable operation of agent systems seems to be a prerequisite for wide-scale adoption of the agent-oriented approach. Literature sketches two major directions for search for a solution: social-level characterization of agent systems (more or less studied) and ontological approaches to agent based system configuration. This paper described our vision towards policy based system configuration.

In the follow-up project called Smart Semantic Middleware for Ubiquitous Computing (UBIWARE) 2007-2010, we attempt to provide a solution advancing into both directions mentioned and somewhat integrating both. The main distinctive features of the platform are externalization of behaviour conditions and restrictions, i.e. agents access the policies from organizational repositories for correspondent configuration of the system, and utilization of the RDF-based Language.

During the last couple of years, policies have gained attention both in research and industry.

Policies are considered as an appropriate means for controlling the behaviour of complex systems.

They are used in different domains for automating system administration tasks, such as configuration, security, or Quality of Service (QoS) monitoring and assurance. Context-Policy-Configuration approach for creation of intelligent autonomous systems is allowing modifying system behaviour without changing source code or requiring information about the dependencies of the components being governed. The system can continuously be adjusted to externally imposed constraints by changing the determining the policies.

This research presents a policy-based approach for supporting the high-level configuration of systems, integrated into the middleware platform. Policies are high-level, declarative statements governing choices in the behaviour of a system. Our "policy-driven middleware" extends the traditional middleware architecture with an extra layer that hides complexity when possible and enables simplified application development and maintenance by offering the means to express, validate and enforce policies.

ACKNOWLEDGEMENTS

This research has been performed as part of UBIWARE project in Agora Center (University of Jyvaskyla, Finland) funded by TEKES and industrial consortium of Metso Automation, ABB, Fingrid, Inno-W and Hansa Ecuras. We are very grateful to the members of "Industrial Ontologies Group" for fruitful cooperation.

REFERENCES

Ankolekar, A., Burstein, M., Hobbs, J.R., Lassila, O., Martin, D. L., McDermott, D., McIlraith, S.A., Narayanan, S., Paolucci, M., Payne, T.R. and Sycara, K., (2002). DAML-S: Web Service Description for the Semantic Web, URL: http://www-2.cs.cmu.edu/ ~terryp/Pubs/ ISWC2002-DAMLS.pdf.

Berners-Lee, T., Hendler, J. and Lassila, O., (2001). The Semantic Web. Scientific American 284(5), 34-43.

Clabby, J., (2002). Web Services Executive Summary, URL: http://www-106.ibm.com/ developerworks/ webservices/library/ws-gotcha/?dwzone= webservices.

Curbera, F., Dufler, M., Khalaf, R., Nagy, W., Mukhi, N., Weerawarana, S., (2002). Unraveling the Web Services Web: An introduction to SOAP, WSDL and UDDI, Internet computing.

Damianou, N., Dulay, N., Lupu, E. and Sloman, M., (2001). The ponder policy specification language. Proceedings of the 2nd International Workshop on Policies for Distributed Systems and Networks, 2001.

FIPA, (2001). FIPA Interaction Protocol Library Specification Specification, FIPA00025. URL: http://www.fipa.org/specs/fipa00025/

Jennings, N., (2000). On agent-based software engineering. Artificial Intelligence 117(2), 277–296

Jennings, N., (2001). An agent-based approach for building complex software systems. Communications of the ACM 44, 4 (2001) 35–41.

IBM, BEA Systems, Microsoft, SAP AG, Sonic Software, and VeriSign, (2006). Web services policy framework (WS-Policy), March 2006.

Kagal, L., Finin, T. and Joshi, A., (2003). A policy language for a pervasive computing environment. In Proceedings of the 4th International Workshop on Policies for Distributed Systems and Networks, 2003.

Katasonov, A. and Terziyan, V. (2007). SmartResource Platform and Semantic Agent Programming Language (S-APL), In: P. Petta et al. (Eds.), Proceedings of the 5-th German Conference on Multi-Agent System Technologies (MATES'07), 24-26 September, 2007, Leipzig, Germany, Springer, LNAI 4687 pp. 25-36.

Kaykova, O., Khriyenko, O., Kovtun, D., Naumenko, A., Terziyan, V. and Zharko, A., (2005). General Adaption Framework: Enabling Interoperability for Industrial Web Resources, In: International Journal on Semantic Web and Information Systems, Idea Group, ISSN: 1552-6283, Vol. 1, No. 3, July-September 2005, pp.31-63.

Khriyenko, O. and Terziyan, V., (2006). A Framework for Context-Sensitive Metadata Description. In: International Journal of Metadata, Semantics and Ontologies, Inderscience Publishers, ISSN 1744-2621, Vol. 1, No. 2, pp. 154-164.

OASIS, (2005). eXtensible Access Control Markup Language (XACML) version 2.0, OASIS standard, February 2005.

Paolucci, M., Kawamura, T., Payne, T. R., Sycara, K., (2002). Importing the Semantic Web in UDDI. URL: http://www-2.cs.cmu.edu/~softagents/papers /Essw.pdf

Semantic Web, (2001). URL: http://www.w3.org/2001/sw/

Uszok, A., Bradshaw, J., Jeffers, R., Suri, N., Hayes, P., Breedy, M., Bunch, L., Johnson, M., Kulkarni, S. and Lott, J., (2003). KAoS policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement. In Proceedings of the 4th International Workshop on Policies for Distributed Systems and Networks, 2003.

Wohlstadter, E., Tai, S., Mikalsen, T., Rouvellou, I., and Devanbu, P., (2004). GlueQoS: middleware to sweeten quality-of-service policy interactions. In Proceedings of the 26th International Conference on Software Engineering, 2004.