ABSTRACT

Recent expectations regarding a new generation of the Web strongly depend on a success of Semantic Web technology. Resource Description Framework (RDF)\(^1\) is the basis for explicit and machine-readable representations of semantics of various Web resources and enables a framework for interoperability of future Semantic Web-based applications. However it has been pointed out that RDF is not suitable for describing highly dynamic and proactive resources (e.g. industrial devices, processes, etc.). Therefore, an appropriate extension of the existing RDF is necessary. This paper presents the Proactivity Layer of the Smart Resource in Semantic Web with the Resource Agent Behaviour definition. Process performance strategies and coordination methods of such proactive goal-driven resources are considered.

KEY WORDS

Proactive resource coordination, Semantic Web, intelligent scheduling, heterogeneous resources collaboration, multi-agent environment

1. Introduction

The modeling of multi-agent systems and the behaviour of the concrete agents within a system have been significant topics in various domains. The model-driven approach to the design of agent behaviours emerged quite some time ago and initially was based on UML modeling [1, 2]. Later this approach was extended to a level of meta-modeling [3]. As one of the mature UML-based methodologies for modeling multi-agent systems, Agent Modeling Language (AML) can be specifically mentioned [4]. Currently, AML is used in commercial software projects, is supported by CASE tools, and, in the near future, a first version of its specification will be presented to the public for its further development. One of the fundamental formal theories about behaviour in multi-agent systems [5] has been developed and lectured on at the Free University of Amsterdam\(^2\).

Two previous works [6], [7] present frameworks as a logical evolution of the RDF to describe proactive, context sensitive, and goal-driven resources. These are the Context Description Framework (CDF) and the Resource Goal and Behaviour Description Framework (RG/BDF), correspondently. One of the main features of the CDF is an ability to describe context-dependent facts (fact-statements) about resources. At the same time RG/BDF brings a new (additional) vision to resource description. It is a description of a resource mental state. If we consider an agent (software agent) as a resource in the Global Understanding eNvironment (GUN) [8], then we face its believes and desires. Thus, we can describe not just the statements that state the facts, but also goals-statements, that state wishful for resource (agent) state of environment, other resources states and etc. Additionally RG/BDF gives us a possibility to describe the rules of resource behaviour via specifying the necessary and sufficient conditions, and rules of the environment change.

When we come to the behaviour annotation of a resource (agent) and the resource proactivity performance stages, we face two challenges. At first, we need a handy and intelligent user interface for the resource behaviour (rules) and the resource mental states specification. And second, we need an engine to run these rules and to perform the actions. These aspects are considered in the chapter #2, which describes the Proactivity Layer architecture of the Smart Resource. Based on such a platform of interactive, proactive resources, many different processes can be modelled. They can be both complex processes (business processes, enterprise integration, distributed maintenance, distributed diagnostic and learning, supply chain management, etc.) and more primitive (personal agents’ interaction, home devices interaction, etc.). It brings a new challenging task of distributed parallel processes coordination. The chapter #3 considers the issues of such coordination of the proactive goal-driven resources. Applying new technologies should not make life (business processes, etc.) complex, but rather should make it easy, flexible and scalable. New technologies should be simple and attractive for users, with a purpose to utilise such kind of multi-agent system.

\(^1\) http://www.w3.org/RDF/
\(^2\) http://www.vu.nl/
2. Proactivity Layer architecture of the Smart Resource

The approach to heterogeneous resource integration in the Semantic Web has been presented in earlier works [9]. Accordingly to this approach, we have to supply each resource with an OntoShell - a resource shell with the Adaptation Layer and the Proactivity Layer. In this paper we will accent the resource proactivity and consider the Resource Behaviour Engine and proactive resource coordination. However, firstly we should define the type of data used as an input for the engine. There are several statements (fact, mental, rule and behaviour) that will guide the discussion:

- **Fact Statement** describes the facts of the entire system (environment), including the states of the resources and their sub-histories. The RS/CDF SR_Statement [10,6] (enhanced with a context extension) fits this purpose very well.

- **Non-Fact Statement (Mental Statement)** allows the possibility to describe not just a history as facts, but also to describe the mental states of a resource (accordingly to BDI [Beliefs-Desires-Intensions] Model). According to [7], the Goal_Statement describes a goal that the resource agent is aimed to make true. Let us define all Non-Fact Statements by rgbdfs:NF_Statement (supper-class of rgbdfs:Goal_Statement). As a RS/CDF Statement, this NF_Statement is enhanced with a context extension, but takes on the FALSE-value until the same RS/CDF Statement appears as a Fact Statement in the resource history and then takes on the value of TRUE.

- **Rule Statement** can be referred to the Non-Fact Statements as a statement allows one to describe the rules of the environment modification. Now we can define a statement’s truth dependence on other statements (the Non-Fact Statements), which dictate the necessary and sufficient conditions. Thus, the Rule Engine, which follows these rules, can modify the content of the History (Fact Statements), the Non-Fact Data Storages, and Rule/Behaviour sets in the Operational Memory of the Engines. The RG/BDF Rule Statement gives the possibility to describe a rule in an "AND-OR-NOT" notation that can be easily presented in “AND-NOT” notation as well (see Fig.1). The rgbdfs:trueInContext property plays the role of the logic “AND” operation and the rgbdfs:falseInContext the “AND-NOT” operation; the NF container (connected via the rgbdfs:trueInContext property) is a collection of the productions combined via the “AND” operation; from the opposite site, the NF container (connected via the rgbdfs:falseInContext property) is a collection of the productions combined via the “OR” operation. In the case of multiple usage of the rgbdfs:trueInContext and rgbdfs:falseInContext properties, they are combined via the “OR” operation. Such a Rule Statement can be used for a meta-rules definition as well. The state of a rule is described via the rgbdfs:ruleState_is property, where the values for the rule state is restricted by the rgbdfs:Active and rgbdfs:Passive values. Thus, the NF Statement (which defines the state of a rule) can be activated/deactivated via the Rule Statement and can play the role of a sufficient condition for the subject rule. The meta-rule definition gives a possibility to define the context for the rules and behaviours.

- **Behaviour Statement** is also a statement for rule definition (subclass of Rule Statement). It describes a rule of behaviour performance: fragmentation to more simple behaviours via the rgbdfs:has_Behaviour property or the performance of concrete executable modules (Action) via the rgbdfs:execution property. Analogically with Rule Statement, the activation of the Behaviour Statement depends on the Environmental and Resource Mental (Non-Fact) States.

Thus, the architecture of the proactive layer of the Smart Resource Platform contains four storages: the Environmental (containing the Fact Statements), the Resource Non-Fact States (containing the Non-Fact Statements and Rule Statements) Storages, a storage where ontology and all instances (Resources such as Devices, Services, Human Experts, Agents, etc.) are located, and storage of the programmable executable modules. In reality, the storage of the Fact Statements is presented by two storages: the Operational Memory and the Long Term History. Operational Memory contains updated relevant to performance data. For example, if a statement that the ResourceAgent plays some new role goes to the Operational History, then statement about the previous role should be removed to the Long Term History, or the irrelevant alarm statements should be removed. Such a filter should not allow contradiction within the operational data. And if we try to generalize this approach, we can see that even the mediation platform, which plays the role of host for the resources, can be considered as a Smart Resource with its own proactivity layer. Utilizing such centralization approach, we can create alliances of resources with internal “rules of the game”. On the other hand, cooperation between resources can be realized via the P2P connections.
From the system usability point of view, all of these complex models of element interaction and the functionality of the engines should be hidden from end-user via a handy and intelligent SmartInterface. The main information that a user should specify during a ResourceAgent setting is the goal (a Goal Statement of ResourceAgent aim). User should specify the input data (not necessarily existing fact from the History [if there are not any statements yet], but specify a template – statement without an object). During all of these manipulations, the interface should provide the user all the available information from the ontology and data, which is stored on the Platform: a list of instances, a list of intellectually filtered properties, etc. Thus, if there are semantic profiles of accessible executable modules and web services (with semantically annotated inputs and outputs), then the “behaviour modelling module” on the Platform will generate the behaviour rules automatically (and will try to build process execution). Otherwise, we will need to specify a semantic profile for all of the available executable modules on the Platform and for the web services that will be used. If there is no any executable module or web service which can exactly satisfy the goal, then the goal can be divided to a set of sub-goals based on correspondent information in the ontology or iterative process of automatic sub-goals generation (based on the required inputs for modules that can reach the goal, but necessary inputs are not provided).

Thus, the goal will be reached by using a set of interactive executable modules. Again, if such platform applies for a new infrastructure, it brings the need to define not just the rules of the Agent behaviours, but also the rules of the Environment (whole system) influence (the rules of behaviour context). All these actions the user should do in a worst case, when he/she adapts the platform for a specific case (specific domain). But there is also an easier way to configure the ResourceAgent, if it used for a widespread, widely used, and known process. It is based on the Agent Role specification only, and implies that the ontology contains all the relations between the agent roles and the goals with the correspondent behaviour rules. But, both of these ways assume that a large amount of hard work has been done by ontology engineers beforehand, and that the ontology contains enough knowledge to allow the SmartInterface to demand as little as possible from user.

3. Proactive resource coordination

The main feature of the SmartResource Platform is process performance via Resource Agents communication. All the Platform Agents are designed in a common way to provide interoperability and common approach. General SmartResource Agent Architecture is represented in Fig. 2. The main functionality of the Agent is based on performance of a behavior (set of behaviour-rules) that corresponds to assigned role. The behaviour description is a RG/BDF based script, which can be loaded from the ontology of the Roles. Based on RG/BDF Script-Role and RS/CDF based beliefs descriptions, Agent runs reusable atomic behaviours – Actions (executable modules). Action performance results change of Environment State. In another words, this performance modifies Agent beliefs. Such atomic behaviours can be downloaded from remote pool of atomic behaviours on demand. But basic set of them and frequently used Actions can be placed locally on the Agent platform. The basic set of Actions that recently used in current prototype of SmartResource Platform is represented in Fig. 2.

The General Networking Framework (GNF), as a part of SmartResource project area of interests, considers an opportunity of ontological modeling of business processes as integration of component behavioral models of various business actors in such a way that this integration will constitute the behavioral model of an agent responsible for the “alliance” of the components. This means that such “corporate” agent will monitor behaviors of the proactive components against the constraints provided by the integration scenario. Such model is naturally recursive and this means that the corporate agent can be a component in a more complex business process and will be monitored itself by an agent from the more higher level of hierarchy. Before we start talking about process integration issues, let us answer the question: What is a process in GUN [8]? Each resource in dynamic industrial World is a process and each process in the World is a resource. Process – is similar resource to other resources in GUN (Device, Service and Human/Expert), but does not belong to the world of physical resources. As all GUN resources, Process has own properties that describe Process’s state, history, sub processes and belongingness to upper-process.

Fig. 2 - Agent Shell architecture.
Following the principles of GUN resource, each Process is enhanced with an Agent that serves Process as a resource and actually realizes it as a behavior engine. Each process is a sequence of the actions (rgbdfs:Execution) that results in achievement of the final goal. So, each Agent per se is a process. In this case Agent Behavior plays role of a sequence of the actions and final result is represented by Agent Goal. Each GUN resource can theoretically be involved to several processes, appropriate commitments and activities, which can be either supplementary or contradictory. This means that the resource is part of several more complex resources and its role within each of the resource might be different. There are some models of upper-process organization. But before we start consider the models, we should state some definition. Let us consider executable module as an atomic non configurable actions. Thus, the choreography of a subject resource by its Agent via action performing is a non-configurable atomic leaf-process. In this case, Agents behave accordingly to certain plan – planned set of behaviours. But, such simple processes can be organized in alliances – Process. The main function of a Process-Agent is the orchestration of a set of sub processes. Following this approach, architectures of arbitrary nested processes can be built, where leaf-processes are physical world Resource-Agents (Device-Agent, Service-Agent and Human/Expert-Agent).

One aim of Process (upper-process) creation is to organize cooperative work of sub processes for improving their individual performance. Each Agent should be supplied with a behaviour-planer module that generates plan for behaviour performance without any conflicts. And in this particular case, Process-Agent should utilize behaviour-planer to build plan of sub processes cooperative work and set constraints on their own plans. Another aim Process creation is to utilize other processes to reach another separate, let us say - group-goal. In this case, achievement of the sub processes’ goals depends on commitments and contracts between all parties. Thus, Agent-owner of this group-goal plays two roles: role of the sub process as another sub processes in this Process (with one difference – it has just goal and does not have atomic behaviour) and role of Process-Agent that performs orchestration of the sub processes. If we separate these two roles, we come to first model where we have blank sub process (has just goal and does not have any atomic behaviour) among sub processes, but achievement of this group-goal takes biggest priority. Figure (Fig. 3.) shows us generalized model when Process-Agent replans sub processes behaviours accordingly to sub processes goals achievement priorities. Nobody can guarantee stability of an environmental data if data space is shared among several Processes. It brings a need to replan the behaviour depending on the changes. The optimal way to reduce amount of replans is to collect all Processes that share same data space under one upper-process, if it is possible. Generally, all the behaviours are represented by the set of rules that operate with the classes of resources (not the concrete instances). But during the behaviour processing by Behavior Engine all the rules are bounded with concrete instances. After such bindings we may have the conflict situations. If two processes use different instance spaces (spaces of facts, desires and etc.), then no conflicts may happen. But, if they share the same instance space, they can block others process performance by changing the shared information space. Actually, while those Resource-Agents are living separately, no one cares about this conflicts of performance and they are concentrated just on achievement of the own goals. But when those two processes are members of another bigger upper-process, the duty of the Process-Agent is to resolve the conflicts via setting the constraints for behaviours of its members to reach the own goal and goals of the members. Initial behaviour of Process-Agent contains such set of actions as: collection of all the behaviors of process members and convert them to the set of rules; applying an algorithm to build a sequence of actions (performance plan) for optimal achievement of a final goal and intermediate goals (if necessary) based on behavior-rules of sub processes; setting the constraints on behaviours of the members for conflict situations (when several rules may be applied, but result the different states – Environment State). In another words, we have a need to define and provide the meta-behavior-rules for the sub processes. Such constraints for process behaviour-rules change behaviour of the Resource-Agent and restrict the degrees of an Agent freedom. Actually with its degree of freedom sub process sacrifices to upper-process when becomes a part of it. It is not necessary, that it negatively affects sub process’s goal achievement, but often the opposite – it can result to speedup of the goal achievement.

Let us consider an example that explains some case for process integration. For the easiest rule representation we will use Production Model of knowledge representation. But we should remember that Agent operates with the RG/BDF behaviours, not with the rules. RG/BDF behaviour is a subclass of RG/BDF rule and has reference to Execution (Executable module) in the right part. In turn, each Execution (action performance) results to certain changes in the Environment State. Further, in figures, we convert the behaviours to the rules for easier explanation. In figure (Fig. 4(a).) we can see two sets of rules that are behaviours of two separate Agents. Each of them has own goal sub state of the Environment (a sub set of Environment statements) and shares the common State of Environment. Numbers in the circles show the order of rule applying for each rule set to optimally achieve the correspondent goal. Also from the arrows you can see the rules that can be applied at the same time for current state of the Environment. Now, from the figure (Fig. 4(b).) we can see rule set of the process that is an upper-process for previous two processes. This rule set is a combination of the sub-processes rules. Also, we can see the final goal of this upper-process and order of rule applying. This order also is shown in figure via numbers in squares.
And now we can see that rule orders of sub processes and order of upper-process are different. This is because upper-process is aimed to resolve the conflicts between sub processes and organizes their cooperative work. The rule order \{ R^1_1, R^1_2, R^1_3 \} of Agent_1 is the optimal one to reach the goal. Rule R^1_3 will be applied first, because the rule R^1_2 results stop of the process. This is because P_1 and P_4 cannot be achieved any more until some another process changes the Environment State with P_1 and P_4. But for the upper-process, the rule R^1_1 should be applied first, because the rule order \{ R^1_2, R^1_3, R^1_4 \} is optimal order for conflict resolution and achievement of all the goals. Also the rule R^1_4 should not be applied ever, because P_8 will not be achieved and it results stop of the process (Agent_3).

Thus, the main functionality of the upper-process is to define the rule constraints for sub processes with the aim to realize orchestration of them. With all this we come to meta-rules for Agent’s Behaviours. Figure (Fig. 5.) shows us meta-rule enhanced Agent Behaviours (Process_1 and Process_2). Now the rule order of upper-process is determined by the constraints of sub processes’ behaviour-rules. But as we mentioned before, Agent operates with the RG/BDF Behaviours, not with the rules. In this case, meta-rule enhancement means Agent behavior-rules set extending with additional behaviours that plays role of meta-rule and switch the behavior-rules conditions. Regarding this process coordination approach, upper-process Agent should provide additional behavior-rules (meta-rules) with the necessary RG/BDF Executions (atomic executable modules - Actions) that perform a behaviour-rules condition switching. It makes sense to define rgbdfs:RuleConditionSetter as a subclass of rgbdfs:Execution and supply this class with two properties: rgbdfs:subjectRule and rgbdfs:subjectRuleCondition. Action, that changes rule condition, gets as an input certain instance of rgbdfs:RuleConditionSetter class and references to subject rule (its condition should be set) and condition value itself. As a result, correspondent Fact Statement (about the rule condition) will be added to Active Data Space. Thus, the RG/BDF behaviour-rule description approach fits very well the constraints definition via adding a restriction behaviour statement.

4. Conclusion

This paper has presented the Proactivity Layer of a Smart Resource Platform, and has described the components of the platform that are based on the successful beneficial extensions of RDF. This approach provides a semantically enhanced way for the rule and meta-rule definition. Again, each industrial resource can theoretically be involved to several processes, appropriate commitments and activities, which can be either supplementary or contradictory. This means that the resource is part...
of several more complex resources and its role within each of the resource might be different. Some models and techniques for coordination of distributed dynamic proactive goal-driven resources, as the processes, have been presented in this paper. The ontology-driven approach in toward modeling agent behaviour as a context-sensitive dynamic change of standardized and reusable roles, goals and actions, is anticipated to become a powerful solution for providing some benefits compared to conventional model-driven approaches.

Acknowledgments

This research has been performed as part of the SmartResource project at the Agora Center (University of Jyväskylä, Finland) and is funded by COMAS, as a part of my doctoral study. I am grateful to the members of “Industrial Ontologies Group” for fruitful cooperation.

References


