Decision making for industrial agents in Smart Grid applications

Gulnara Zhabelova[1], Valeriy Vyatkin[1,2], Viktor Dubinin[3]
[1]Department of Computer Science, Electrical and Space Engineering, Luleå Tekniska Universitet, Luleå, Sweden
[2]Department of Electrical Engineering and Automation, Aalto University, Helsinki, Finland
[3]University of Penza, Penza, Russia
e-mail: gulnara.zhabelova@ltu.se, vyatkin@ieee.org, victor_n_dubinin@yahoo.com

***Abstract*— Agent technology in the power system domain is the realm of theory and laboratory simulation. Agent-based systems, due to their complexity of concept and design, may appear difficult to implement and maintain, and hence more expensive, which is a prohibiting factor against using the new technology. Designing a decision-making system for an agent is one of the most complex tasks in the development of agent-based systems. Designing a decision-making system enabling proactive behaviours while maintaining reasonable complexity is a challenge. This paper proposes an deliberative decision making system for an industrial agent which allows pro-active behaviour of the agent while maintaining reasonable complexity. It employs a combination of practical and procedural reasoning principles. The formal definition of the proposed deliberative layer is derived in the paper. The procedural reasoning is based on IEC 61850 Logical Node concept and implemented with IEC 61499 reference architecture. The proposed agent architecture is demonstrated on the Fault Location Isolation and Supply Restoration application and validated via so-simulation with Matlab model of the sample distribution network.**

Keywords*— Agent, Smart Grid, Reasoning, Decision making, IEC 61499, IEC 61850, deliberation, power system automation, distributed grid intelligence.

I. INTRODUCTION

A Smart Grid is a complex energy infrastructure managing distributed renewable energy generation, storage, transmission, distribution and consumption. The Smart Grid allows for bi-directional flow of energy, enabling consumers to sell excess energy back to the grid.

To control such a complex and highly distributed infrastructure, the Smart Grid has to employ new generation distributed automation and control systems, i.e. Distributed Grid Intelligence (DGI). DGI is a network of distributed nodes performing intelligent control to achieve local goals and participating in overall Smart Grid operation and control to achieve system objectives. These nodes are essentially the agents operating autonomously, reacting on the environment and proactively negotiating among themselves to achieve the system objectives, exhibiting social behaviour.

However, the majority of agent-based systems applied to power system automation domain are the laboratory simulations. Due to the nature of agent technology, the designed system can exhibit a wide variety of expected and unintended behaviours, which may lead to inefficient or unstable operation [1, 2]. Agent-based systems, due to their complexity of concept and design, may appear difficult to implement and maintain, and hence more expensive, which is a prohibiting factor against using the new technology [2].

Designing a decision-making system for an agent is one of the most complex tasks in the development of agent-based systems. The complexity of the decision-making can be reduced by using the reactive architecture. These agents simply respond to their environment [3, 4]. Reactive architecture directly links the sensory data to acting capabilities of an agent [2]. However, it is difficult to implement pro-activeness and goal-oriented behaviour based on reactive architecture alone. The architecture only looks at the present situation and does not take into account long term goals. This is where the challenge lays, that is, designing a decision making system enabling proactive behaviours while maintaining reasonable complexity.

The decision-making system, which allows for pro-active behaviour is a deliberative architecture. In the deliberative architecture the goals and plans are explicitly represented. The most popular deliberative agent architecture is the Beliefs, Desire and Intentions (BDI) of Rao and Georgeff [3]. The beliefs are the sensory input accumulated over time; desires are the delegated goals. Based on its beliefs and desires, an agent decides on its intentions, i.e. what it wants to do in the future. The intentions direct the actions the agent is committed to perform, unless it has to abandon the intention, given that the situation has changed.

The BDI agents can pursue their goals pro-actively and react to the environment. BDI architecture is capable to implement autonomous reactive and pro-active agents [2].

The disadvantage of deliberative reasoning is the time needed for an agent to evaluate the updated beliefs and form an intention, i.e. it reacts only after the sensory data has gone through all the steps. For complex agents it may take a relatively long time, and for highly dynamic environments it may be too long to react to changes [2]. Additionally, such deliberation process requires computational resources to be available.

In order to be applied in the power system automation domain, the agent architecture should be practical. That is, to be implementable by a domain engineer, reflect specifics of decision making in the power system automation domain, satisfy system requirements such as reaction time, and be executable on field devices.

This paper proposes a deliberative decision making system for an industrial agent and aims at addressing above mentioned issues and allow reactive and pro-active behaviour of the agent while maintaining reasonable complexity. The proposed agent architecture is based on industrial standards IEC 61499 and IEC61850 and is intended to facilitate industrial adoption of agent technology to the power system automation domain.

This paper will focus on the deliberative layer of the proposed architecture; full description of the architecture can be found in [5].

The paper is structured as follows. Next section II presents a discussion on agent reasoning and describes deliberation process based on practical and procedural reasoning. In section III, the proposed reasoning system for industrial agent is presented. This section derives a formal description of the reasoning process. The proposed decision making system was implemented using IEC 61499 and tested on Fault Location Isolation and Supply Restoration scenario. The implementation is discussed in section IV. Chapter V presents the results. The paper conclusion is outlined in section VI with the overview of the future work.

II. AGENT REASONING

A. *Discussion on agent reasoning: deductive, practical and procedural*

The "traditional" artificial intelligence uses logical reasoning to decide what to do. The logical reasoning operates over *symbolic* representation of the environment and its desired behaviour. This type of decision making is a *deductive reasoning* [4, 6].

It operates with pure logic specifications. Deductive reasoning uses symbolic representation of the world around it, often expressed in formal logical languages such as Description Logic [7]. The reasoning about the world is logical deduction or theorem proving. An agent's decision making is implemented as a logical theory, and then selection of an action is a proof [3, 8]. Deductive reasoning has two key problems: transduction and representation/reasoning. The first problem is translating the real world environment into an accurate adequate description of the world, in time for the description to still be useful. The second problem is to convert the description of the world into symbolic representation and let the agent reason with it in time for the result of the reasoning to be useful.

Logic-based decision making is elegant and has clear semantics [3, 8]. However, it has many disadvantages [3, 8]. Deductive reasoning inherits computational complexity of theorem proving. The assumption of calculative rationality, that the world will not change while the agent is reasoning and the selected action, once decision making is completed, is still rational, making agents based on deductive reasoning not fit for complex, dynamic physical environment [3]. Computational complexity of this decision-making requires both time and resources to be available. Moreover, accurate symbolic representation of the real world, especially sensors, is complex and time consuming. It raises practicability issues of purely deductive reasoning agents [3, 8].

Practical agents and especially industrial agents operate under resource constraints of field devices such as fixed size memory and fixed processor performance. These limitations restrict the size of computation, which the agent can perform. The agent cannot deliberate forever, in an industrial environment there are time constraints of some kind. That means that deliberation should be carried out within a finite number of processor cycles. Thus the reasoning should be effective and decisive. One such method is *Practical Reasoning* [3]. This reasoning process involves two parts: deliberation and means-end reasoning. Deliberation resolves what state of affairs the agent wants to achieve. Means-end reasoning is a planning process, deciding how to achieve the desired state of affairs. The agent, through the deliberation process, will generate intentions, and by means-end reasoning perform planning and arrive to a plan of execution.

Deliberation and means-end reasoning are computational processes. Resources and time are spent on generating desires, re-considering intentions and generating plans of actions. As mentioned above, industrial agents work under both time and resource constraints. The actions, which an industrial agent can take in many cases are known, restricted and designed in. Thus, the plans of the agents can be pre-compiled. This, in principle, removes the need for planning processes in the agent's reasoning. The agent's purpose is also known, which defines the desires of an agent, and thus eliminates option generation from the reasoning process. The objectives or desires of an agent and all possible plans are provided at compile time as a library. In this manner, the required processing resources can be reduced. Firstly, by an abridged deliberation process, which is still sufficient for intention filter function; and secondly, by substituting means-end reasoning with a library of pre-compiled plans. The additional process of plan selection is introduced.

This discussed alternative architecture is so-called *Procedural Reasoning System* (PRS) [3]. PRS does not include a planning process. Instead it provides a library of pre-complied plans. The PRS is the basis of the deliberative part of the proposed architecture. Since the agent is based on an IEC 61850 Logical Node (LN, described in section III) and the LN defines the agent's beliefs and predefines the agent's purpose and desires. The domain specific function of an LN influences the possible actions of the agent, which will ultimately be designed in by the domain engineer. Thus, the agent will have a library of plans, predefined desires and formed (structured) beliefs. These are the building blocks of the PRS.

PRS implements the BDI architecture (Belief, Desire, Intention) and provides the structures of the BDI concept [6]. Procedural reasoning contains three main predefined data sources: plan library, beliefs structure and desires. The deliberation process is the process of selecting between different possible competing plans. This reasoning engine is referred to as the *interpreter*. The interpreter will consider beliefs, desires, possible plans and current intentions to select a plan, which will realize the desired state of affairs.

A plan consists of a goal, a context and a body. A goal is a post-condition of a plan, the context is the precondition and the body can be both a set of actions to carry out, or goals to achieve. An agent typically will have a top-level goal. The interpreter maintains goals to be achieved in the *intention stack*. Current intention is on the top of the stack. The options are selected from possible plans, where the post-conditions match the goals placed on the top of the intention stack, while preconditions are satisfied by the current beliefs. Then the interpreter filters out the plans to execute by using either meta-level plans or utility measures of some kind. The chosen plan is executed, and it results in the execution of actions and possibly in new goals pushed into the intention stack. If the plan fails to achieve the goal, the agent selects different plan.

B. Proposed deliberation process based on practical and procedural reasoning

The general BDI architecture control flow is depicted in Fig. 1[9]. There are three main steps: deliberation, planning and plan execution. The deliberation process decides what to achieve by generating possible goals or intentions and filtering the best intention for the agent to commit to. Means-end reasoning performs the planning algorithm, which generates a plan. The control loop starts from belief revision updating them with sensory inputs. Using beliefs and current intentions, the option generation function provides a set of achievable goals or intentions. These goals pass through the filter to select some from among the competing intentions. Now, the agent is committed to achieve them. Current beliefs and intentions are the input into the planning process, which results in a plan. If the plan is sound, then the agent can execute the planned actions.

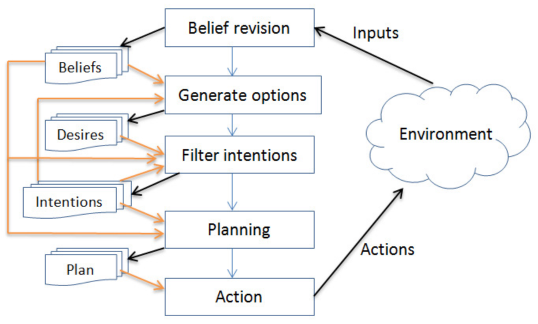

Fig. 1 Control loop of deliberative model.

In the procedural reasoning, there is no planning in principle. By establishing the agent on the basis of well-defined LN, the purpose of the agent, its desires, intentions and possible actions are known at the design stage. Therefore, this allows design of a library of possible and sound plans. The pre-compiled plans can simplify the deliberation process and remove means-end reasoning. The option generation function is simplified to a match and select, rather than generating options/goals from scratch. Filter function is stripped down to the selection of the suitable plan comparing their utility ratings, priorities or using a meta-level plan. This results in agent decision making being simpler, faster and less demanding on resources.

In the next section, a formal model of the proposed agent architecture is derived based on the formal model of a practical reasoning BDI architecture, and procedural reasoning principles.

III. PROPSOED PROCEDURAL REASONING SYSTEM FOR AN AUTOMATION AGENT

The agent architecture proposed in [5] is based on industrial standards and is aimed at facilitating industrial adoption of agent technology to the power system automation domain. The intention of this section is to describe the deliberative decision making of the agent and derive its formal definition.

The foundation of the proposed agent is the IEC 61499 and IEC 61850 standards.

The IEC 61850 standard "Communication networks and systems for power system utility automation" [10] reflects the advanced industrial practice for designing substation automation, control and protection applications. IEC 61850 decomposes the power substation, including functions for monitoring, control and protection, down to objects so-called LN. An LN describes a domain specific automation function as its information model. For example, a function overcurrent protection is modelled as PIOC LN.

The agent is based on an LN, so that an LN is mapped to an agent according to its objectives and functions. An agent inherits the information model of the LN as its beliefs and the domain specific function of an LN as its desires or objectives. The beliefs and the desires will define the intentions of the agent. Therefore the LN information model and modelling function will determine actions of the agent.

Beliefs are the sensory inputs coming from the environment, communication data coming from network and current internal context of the agent. LN is the structured data of the respective domain specific function. This information composes the beliefs of the agent. Let us denote B_ln be current beliefs of the agent. Let Bel_ln be a set of agent's beliefs, then $B\_ln \subseteq Bel\_ln$. Beliefs are updated with perceptual inputs using the belief revision function:

$$brf: Bel\_ln \times 2^{Per} \rightarrow Bel\_ln$$

Per is a non empty set of percepts.

For instance, beliefs are updated with environment perception, which includes also communication data, using above defined function as $B\_ln \leftarrow brf(B\_ln, Per)$.

Desires describe the purpose of the agent or top level goals. They are not directed on actions, they determine the main course of the agent, which influences its intentions. The respective domain specific function of the LN defines the desire of the agent. Let us denote desire of the agent as D_ln.

Intentions are practical goals whose successful achievement will lead to achievement of the desire, i.e. top-level goal. Intentions are persistent goals, which are directed towards actions. An intention is realised by executing a set of actions. In proposed architecture, the intentions are described in the plans, as post-conditions, i.e. the goal to be achieved by executing the plan. So then an intention becomes a goal to be accomplished. Let us denote I_ln as a current intention and Int_lnInt^{ln} as a set of intentions, $I\_ln \in Ibt\_ln$

The repertoire of possible actions that the agent based on LN, can take, is known. The actions will be aligned with the purpose of the agent. Let α be an action to be executed, and $Ac\_ln = \{\alpha_1, \alpha_2, \dots, \alpha_n\}$ be the set of all possible actions of the agent. A plan in this architecture is a sequence of actions $\pi = \alpha_1, \alpha_2, \dots, \alpha_n$, where $\alpha_i \in Ac\_ln$. Then $Plans\_ln$ is a set of all plans designed for the agent (a library of plans) $Plans\_ln = \{\pi_1, \pi_2, \dots, \pi_n\}$.

The deliberation is represented by two functions: option generation and filtering. An option generation function selects the possible plans to achieve, given intention and current beliefs. This set of possible plans is denoted as $P, P \subseteq Plans\_ln$. The function is defined as follows:

$$options: 2^{PLans\_ln} \times Int\_ln \times Bel\_ln \to 2^P$$

So then using $P \leftarrow options(Plans\_ln, I\_ln, B\_ln)$ an agent generates options (P) for achieving intention I_ln. From the generated set of possible plans capable to achieve current intention, P, a filtering function selects one plan π_i according to the defined criteria. This deliberation process can take various forms. Some of them simply based on such criteria as priority or utility ratings, and others use meta-level plans. Meta-level plans are plans about plans and can change the direction of agents' intentions. Let us assume, that the plan is selected based on utility rating, which is expressed with a number from a set of natural numbers ℕ. Then, we denote filtering function as

$$filter: 2^P \times \mathbb{N} \to P$$

For example, $\pi \leftarrow filter(P, \mathbb{N})$ selects a suitable plan π from the generated options P.

The agent maintains intention stack with the goals or intentions pending to be accomplished. The intention stack is a linear ordered set and denoted as $Stack = \{I_1\_ln, I_2\_ln, \dots, I_n\_ln\}$, where $I_i\_ln \in Int\_ln$.

To manipulate plans and intention stack, there is need to define the following auxiliary functions. If M denotes a linear ordered set $M = \{m_1, m_2, \dots, m_n\}$, then

- *hd(M)* is the top most entry in the set - m_1. The procedure does not delete the entry from the set, it copies it over to the variable, e.g. $i \leftarrow hd(M)$ implies that $i = m_1$ and $M = \{m_1, m_2, \dots, m_n\}$;
- *tail(M)* is the set comprising of all the entries of set M except the first entry, and the order of entries is preserved, e.g. $M_1 = tail(M)$, then $M_1 = \{m_2, m_3, \dots, m_n\}$.

For a sequence of actions of a plan $\pi = \alpha_1, \alpha_2, \dots, \alpha_n$ where $\pi \in P$,

- *execute(α)* is a procedure that executes given action α;
- *empty(π)* is Boolean function indicating whether the π is empty, i.e. there are no actions left to execute in the pan π;
- *head(π)* is first action α_1 in the plan body of π;
- *tailPlan(π)* is the set comprising of all actions of the plan π, except the first action α_1, and the order of actions is preserved.

The agent maintains an intention until it has been succeeded, the intention is impossible or all planned actions are executed. Therefore, corresponding functions are defined as follows. A function evaluating whether the intention has been achieved is defined as

$$succeeded: Int\_ln \times Bel\_ln \times P \to \{true, false\}$$

The function $succeeded(I\_ln, B\_ln, \pi)$ considers plan $\pi \in P$ and the current beliefs $B\_ln \in Bel\_ln$ and decides whether the intention $I\_ln \in Int\_ln$ has been satisfied. Considering current beliefs $B\_ln \in Bel\_ln$ and plan $\pi \in P$, current intention $I\_ln \in Int\_ln$ is identified as impossible by function $impossible(I_{ln}, B\_ln, \pi)$, which is defined as follows

$$impossible: Int\_ln \times Bel\_ln \times P \to \{true, false\}$$

These two functions represent the commitments to the means, i.e. to the plan. Commitment to the ends, i.e. intention or goal, is realised through function

$$reconsider: Int\_ln \times Bel\_ln \times P \to \{true, false\}$$

It is a Boolean function. Based on current beliefs $B\_ln \in Bel\_ln$ and a plan $\pi \in P$, the $reconsider(I\_ln, B\_ln, \pi)$ function returns "*true*" when it decides, that the intention $I\_ln \in Int\_ln$ needs to be reconsidered by the agent. The reconsideration is the deliberation process: generating options and filtering out an intention to commit to. Fig. 2 presents control loop of the proposed deliberative model. The desire is pushed into the intention stack at the start up. This will motivate the agent to operate and generate new intentions. The rest of the algorithm is the loop.

The loop starts updating the beliefs with the perceptual inputs with function $see: E \to Per$, where *Per* is a set of perceptions and *E* is a set of states of the environment. The intention to be achieved is always at the top of the stack. The agent gets the intention I_ln from the stack, and decides how to achieve it. This will result in selection of a plan to commit. This plan is then executed. The procedure *execute(α)* performs a given action $\alpha \in \pi$. The action α may push new intentions into the stack.

The agent implements *single-minded commitment* with *while* loop in rows 10-21. The *while* loop is dedicated to a chosen plan and will continue until there are no actions in the plan to execute or the intention has been succeeded or become impossible. The agent is reasonably *open minded.* The agent maintains an intention as long as it is still believed necessary to achieve. The deliberation is resource and time demanding process, so it is better to deliberate only when it is necessary. After executing an action of the selected plan, agent updates its beliefs and updates current intention with the intention on top of the stack (row 16). If action of the plan did not push new intention into the stack, then the current intention will not change. Next, $reconsider(I\_ln, B\_ln, \pi)$ function (row 17) checks if the intention, the current plan is aimed for, is still valid, i.e. there is still a reason to achieve the intention. If it decides the current intention (lets denote it as I_j_ln) should be reconsidered, an agent generates new options and filters out a new plan. And the *while* loop 10-21 will start executing the new plan. A new plan can be selected in two cases. In the first case, new intention I_{j+1}_ln has been pushed in to the stack, and therefore $reconsider(I\_ln, B\_ln, \pi)$ function decides to deliberate. Deliberation selects a new plan, which is suited for the new intention.

```
1.  B_ln ← B_0_ln;                 /* B_0_ln are initial beliefs */
2.  Stack ← D_ln;                  /* D_ln is the top level goal or desire*/
3.  while true do
4.      Per ← see(E);              /* Get the next percept Per */
5.      B_ln ← brf (B_ln, Per);   /*Update beliefs*/
6.      I_ln ← hd(Stack);           /*Copy intention at head of Stack into I_ln*/
8.      P ← options(Plans_ln, I_ln, B_ln);
9.      π ← filter(P,N);
10.     while not (empty(π) or succeeded(I_ln, B_ln, π) or impossible(I_ln, B_ln, π)) do
11.             a ← head(π);
12.             execute(a);
13.             π  ← tailPlan(π);
14.             Per ← see(E);      /*Get the next percept Per */
15.             B_ln ← brf(B_ln, Per);    /*Update beliefs*/
16.             I_ln ← hd(Stack);     /*Copy intention at head of Stack into I_ln*/
17.             if (reconsider(I_ln, B_ln, π)) then
18.                     P ← options(Plans_ln, I_ln, B_ln);
19.                     π ← filter(P,N);
20.             end_if;
21.     end_while;
22      Stack ← tail(Stack); /*Remove current intention from head of stack */
23. end_while;
```

Fig. 2. An algorithm of the proposed deliberative model.

In the second case, the current intention has not changed (still I_j_ln), but beliefs have changed, which results in a new more suitable plan. If the deliberation resulted in a new plan which is different to the current plan, then the function $reconsider(I\_ln, B\_ln, \pi)$ was optimal and it identified the intention and plan, which were no longer valid. On the other hand, if new plan is the same as the current plan, the function $reconsider(I\_ln, B\_ln, \pi)$ was not optimal and the agent wasted time deliberating.

Next section will describe the implementation of the proposed deliberative reasoning with IEC 61499.

IV. Application of the Proposed Agent Reasoning System in the FLISR Scenario

The proposed deliberation reasoning is implemented with the IEC 61499 reference architecture.

IEC 61499 is designed for development of distributed systems in automation and control. The IEC 61499 standard "defines an open reference architecture for next generation of distributed control and automation" [11]. The key technologies provided by the standard are design structures, such as Basic Function Block (FB), Composite FB and Service Interface FB (SIFB); Application model and abstract Device Model. FB encapsulates automation functions or any programming modules in a portable and re-usable, platform independent form. An event driven processing of IEC 61499 can provide faster reaction time than the cycle based execution of industrial controllers such as Programmable Logical Controllers (PLCs) [11].

Fig. 3 shows the example of the *CSWI* agent from the Fault Location Isolation and Supply Restoration (FLISR) application, described in [12].

There are 2 reactive behaviours: *OperateXCBR* and *FaultIsolatedKeeper*, constituting reactive layer of the architecture.

The deliberative layer is comprised of five plans: ProvideAltertnavieSupply, FaultIsolation, FaultLocated, Restore and GetHelp; an *IntentionStack* and the reasoning engine - *Interpreter*.

The interface of the *IntentionStack* FB allows for plans to push new intentions and the *Interpreter* to read the top of the stack and to delete an intention when necessary. A plan is implemented with a basic FB. Each plan has input *StartPlan* and outputs *PlanIsCompleted* and *newInt* associated with *newIntVar*. Using provided outputs plans can push new intentions into the *CSWI_Stack*. Each plan also has a logic dependent interface for reading current beliefs and outputing actions. The interface of the *Interpreter* provides signals for controlling stack and plans and also reads necessary beliefs in order to deliberate a suitable intention and a plan.

The behaviour of the deliberative layer is controlled by *Interpreter* and depicted in Fig. 4. It implements the control loop presented in Fig. 2. The *Interpreter* requests the intention at the top of the stack and starts deliberating considering

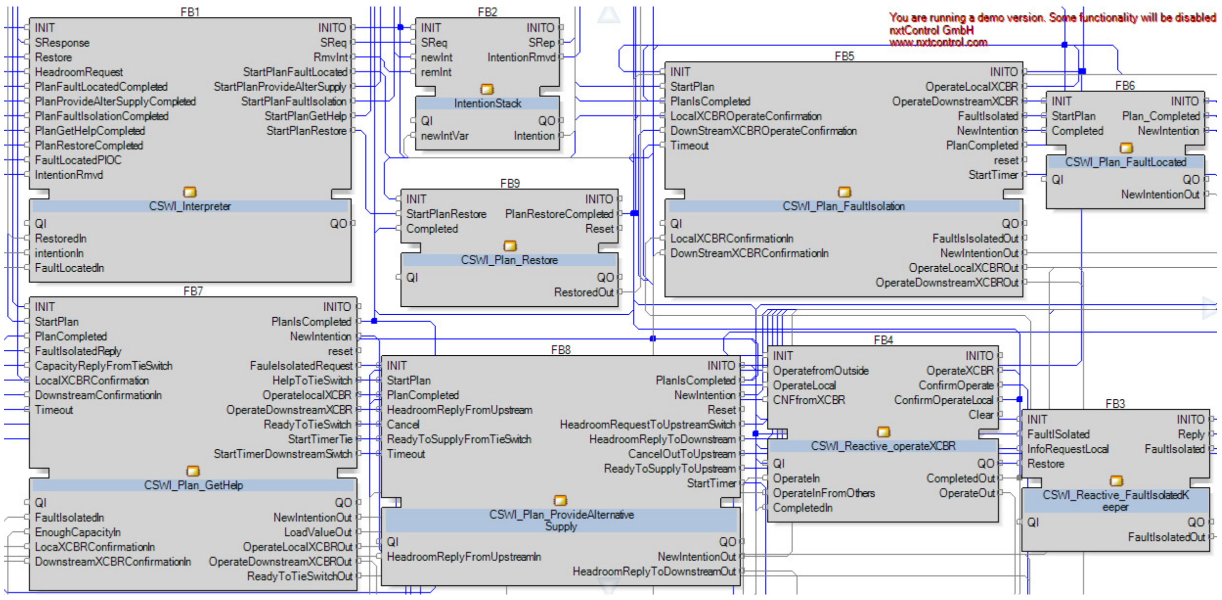

Fig. 3. CSWI agent. Internal structure: reactive behaviours, plans, intention stack and interpreter.

current beliefs. There are three defined intention for CSWI agent: "*NormalOperation*", "*FaultOperation*" and "*Restore*". In these states the current intention is checked if it has succeeded or become impossible, in which case the interpreter deletes the current intention from the stack. If the intention is valid, the agent selects a suitable plan, based on possible options (pre-compiled plans), and current beliefs. The *Interpreter* performs one plan at a time. Once plan is completed, *Interpreter* updates the intention and starts the deliberation considering the updated beliefs.

Under the "*NormalOperation*" intention, available plans are: *FaultLocated* and *ProvideAlternativeSupply*. When the CSWI is on the faulty feeder, then the *FaultLocated* plan is performed, based on the data sent by local PIOC. This plan pushes "*FaultOperation*" intention into the top of the stack. When the CSWI is on the healthy feeder, the *PlanProvideAlternativeSupply* plan is performed, by the request from tie switch or downstream CSWI agent. This plan pushes "*Restore*" intention into the stack.

Under the "*FaultOperation*" intention, there are two plans: *PlanFaultIsolation* and *PlanGetHelp*. When fault recorded is on this CSWI agent's section, then the *PlanFaultIsolation* is selected and fault has to be isolated. When CSWI agent is on the healthy section, it has to find alternative supply via tie switch and adjacent feeder; this is achieved by performing *PlanGetHelp*. By the completion, both of these plans push intention "*Restore*" into the intention stack.

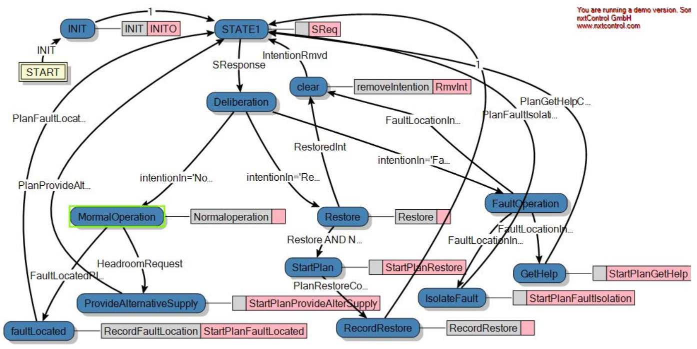

Fig. 4. Deliberation of the CSWI agent. ECC of Interpreter FB.

Under the "*Restore*" intention, there is only one plan - *PlanRestore*, which is initiated on the signal from control centre. This plan restores the state of the agent into the pre-fault state, and once its completed the intention *"Restore"* is achieved and is removed from the stack. The next intention in the stack is "*FaultOperation*" which will be removed from stack as it has been achieved and no longer valid considering current beliefs: the restore signal from the control centre, indicating that the fault has been cleared. On the top of the intention stack remains intention "*NormalOperation*" and *Interpreter* will be back to its original state and deliberate again the necessary plan according to the current beliefs.

The PIOC agent has the same architecture with the own library of plans and reactive behaviors.

The following section will present the simulation set up and the results of the experiment.

V. SIMULATION RESULTS

The above agent architecture in demonstrated on FLISR application, which was developed with the earlier version of the architecture and can be found in [12]. The short description of the scenario is presented below; more details are in [12].

The FLISR agent-based system is developed in nxtControl 1.5. The agent-based FLISR application is coupled with the Matlab model of the distribution system. Agents XCBR and TCTR are interfaced to the circuit breakers and current measurement units correspondingly. Communication is set up using UDP blocks. Matlab simulation runs in parallel with the IEC 61499 agent-based system. More details on the co-simulation framework can be found in [13].

The fault is simulated on section of circuit breaker CB1 (Fig. 5). The task of PIOC agents is to locate the fault and inform corresponding CSWI agents. CB1 must isolate the fault on its section. On receiving the fault isolation event from CB1, switches ROS 1 and ROS2 ask for help corresponding tie switches. Ties switches initiate headroom calculation process on the adjacent feeders Feeder 2 and Feeder 3. When tie switches are satisfied that there is enough headroom capacity on the adjacent feeders, they will signal ROS1 and ROS2. ROS1 and ROS2 will isolate corresponding sections in preparation for receiving alternative supply. Switches ROS1 and ROS4 will close to allow the power flow.

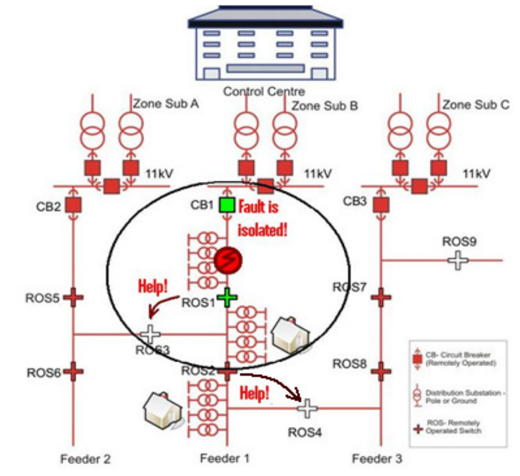

Fig. 5. Sample distribution network. Fault events and search for alternative supply.

The simulation result are shown on the Fig. 6. The fault was cleared in 0.03 seconds of the simulation time after the reclosure. Supply to sections ROS1 and ROS2 was restored almost immediately, in 0.025 seconds of the simulation time. As it can be seen, the sections ROS1 and ROS 2 have restored the supply from feeders 2 and 3 correspondingly. It has to noted, that the simulation time is of a different scale from the real time. The fault clearance time and time taken to restore supply in this experiment is dependent on the computational power of the processor to perform Matlab simulation and run softPLCs, and also on communication delays. What is important in this experiment, is that the agents of the system were able to successfully perform FLISR algorithm in a distributed collaborative manner.

The experiments were conducted simulating fault on sections ROS1 and ROS2, and in both cases the fault was correctly located, isolated and healthy sections restored alternative supply correspondingly.

VI. Conclusion

The paper proposes deliberative decision-making, based on both practical and procedural reasoning. The work aims at reducing complexity and computational intensity of the agent reasoning. It is based on IEC 61850 and implemented with IEC 61499, facilitating an industrial adoption of the agent technology. The developed reasoning system is directly executable on the industrial grade hardware, such as PLCs. Utilising IEC 61850 LN concept facilitates the design process. The agent identification task is straightforward; each LN maps to an agent. Domain specific function of an LN guides the development of the decision making system, defining the agent's purpose, beliefs, possible actions and plans.

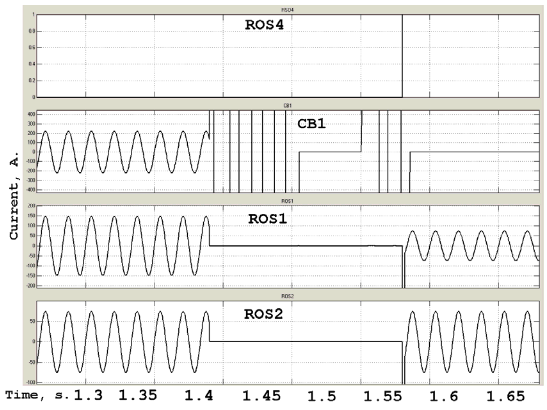

Fig. 6. FLISR simulation results. Feeder 1.

The agent architecture was implemented and tested on FLISR application, where developed agent-based system successfully able to locate the fault, isolate it and restore supply to un-affected sections of the feeder.

The future work involves formalization of agents' beliefs as IEC 61850 OWL ontology. Using reasoning an agent can deduct facts, which are not explicitly defined in its beliefs. As mentioned earlier, the deductive reasoning is not an optimal decision making strategy for the agents operating in the physical environment. Future work can be directed at investigating an efficient deductive reasoning based on the developed IEC 61850 ontological knowledge base.

VII. Acknowledgement

This work has been supported, in part, by FREEDM NSF Centre grant Y3.E.C12.

VIII. References

[1] D. A. Cartes and S. K. Srivastava, "Agent Applications and their Future in the Power Industry," in *Power Engineering Society General Meeting, 2007. IEEE*, 2007, pp. 1-6.

[2] S. Bussmann, N. R. Jennings, and M. Wooldridge, *Multiagent systems for manufacturing control: A design methodology*. Germany: Springer-Verlag, 2004.

[3] M. Wooldridge, *An introduction to Multiagent Systems*. UK: John Wiley & Sons Ltd, 2009.

[4] M. Wooldridge and N. R. Jennings, "Intelligent Agents: Theory and Practice," *The knowledge Engineering Review,* vol. 10, 1995.

[5] G. Zhabelova, "Software architecture and design methodology for distributed agent-based automation of Smart Grid," Doctor of Phylosophy, Department of Electrical and Computer Engineering, The University of Auckland, Auckland, New Zealand, 2013.

[6] M. Wooldridge, *Reasoning about Rational Agents*. Cambridge, MA: The MIT Press.

[7] I. Horrocks, "Description Logic: a Formal Foundation for Languages and Tools," presented at the Semantic Technology Conference (SemTech2010), San Francisco, Claifornia, USA, 2010.

[8] M. Rovatsos, "Lecture 3 Deductive Reasoning Agents," in *Agent Based Systems*, First ed Edinburgh, UK: The University of Informatics, 2012.

[9] M. Rovatsos, "Lecture 4 Practical Reasoning Agents," in *Agent Based Systems*, First ed Edinburgh, UK: The University of Informatics, 2012.

[10] International Electrotechnical Comission, "IEC 61850 Communication networks and systems for power utility automation," ed. 2, Switzerland, 2009.

[11] International Electrotechnical Commission, "IEC 61499-1 Fulnction blocks - Architecture," ed. 1, Switzerland, 2005.

[12] G. Zhabelova and V. Vyatkin, "Multi-agent Smart Grid Automation Architecture based on IEC 61850/61499 Intelligent Logical Nodes," *Industrial Electronics, IEEE Transactions on,* vol. PP, pp. 1-1, 2011.

[13] C. Yang, G. Zhabelova, C. Yang, and V. Vyatkin, "Cosimulation Environment for Event-Driven Distributed Controls of Smart Grid," Industrial Informatics, IEEE Transactions on, vol. 9, pp. 1423-1435, 2013.