Enhancing Distributed Automation Systems with Efficiency and Reliability by Applying Autonomic Service Management

Wenbin (William) Dai *IEEE Member,* Lulea University of Technology, Sweden, w.dai@ieee.org

Valeriy Vyatkin *Senior IEEE Member,* Lulea University of Technology, Sweden and Aalto University, Helsinki, Finland, vyatkin@ieee.org

Victor Dubinin, University of Penza, Russia, victor_n_dubinin@yahoo.com

James H. Christensen, Holobloc Inc., USA, james.h.chrsitensen@gmail.com

***Abstract* – Improvement of flexibility and interoperability is a usual concern of industrial automation systems developers. Service-oriented architecture is one approach promising improvement of flexibility and interoperability in existing distributed automation systems. However, the intelligent self-managing features cannot be fully achieved by just applying the service-oriented architecture. In order to improve efficiency and reliability of distributed automation systems, the service-oriented architecture is extended in this paper by autonomic service management. The design of the autonomic service manager is provided and some key features such as self-configuration, self-healing and self-optimization are demonstrated. The design of a flexible and interoperable execution environment is also illustrated. Some preliminary tests are completed with a case study.**

***Index Terms* — Service-Oriented Architecture, Distributed Automation Systems, IEC 61499 Function Blocks, Interoperability, Flexibility, Autonomic Service Management, Self-* Properties, Ontology, Knowledge Base, Ontology Web Language (OWL), Reasoning, SWRL.**

I. Introduction

Nowadays with increasing complexity and physical size of industrial plants, the developers of automation systems often choose to distribute control logic across several controllers. At the same time the requirements to efficiency and reliability of such distributed automation systems are also increasing. It is expected to be achieved with less or none human effort on account of using more intelligent system management algorithms.

Programmable logic controllers (PLC) are widely used in industrial automation due to their reliability and standardised way of software development compliant with the IEC 61131-3 standard [1]. Another complementary international standard for PLCs, the IEC 61499 standard [2] has been recently released in its second edition. The purpose of this standard is to fill the gap between the IEC 61131-3 standard and distributed automation systems. The key feature of the IEC 61499 standard is event-triggered function blocks which can be deployed across PLCs by simply assigning mapping for each function block. The IEC 61499 compliant PLCs such as NxtControl [3] are more suitable for highly distributed and modulated systems like building automation systems.

IEC 61131-3 PLCs lack of interoperability and flexibility: enabling different brands of IEC 61131-3 PLCs to interoperate is quite difficult. Also changing deployment of existing PLC programs is time-consuming. On the other hand, the IEC 61499 standard provides a higher level of interoperability and flexibility. IEC 61499 function blocks could be easier reallocated from one PLC to another by modifying their mapping to devices. IEC 61499 PLCs from various vendors are able to communicate with each other via standard PUBLISH/SUBSCRIBE function blocks.

However, designing applications with distributed logic is challenging from the perspectives of logic synchronisation. Service-oriented Architecture (SOA) provides a promising solution for reducing this complexity. Because of that it attracts lately attention of many researchers. In [4] it was proposed to implement SOA by means of the IEC 61499 architecture. In SOA-related research such features as self-configuration, self-healing, self-optimisation and self-protection have been addressed by proposing autonomic management mechanisms. Therefore, in order to enhance energy efficiency and reliability of distributed automation systems, a new PLC execution environment architecture with autonomic management is required.

The benefits of shifting to Service-oriented automation design (SOAD) paradigm are tremendous. Self-configuration of PLC program deployment enabled by the SOAD will reduce PLC software development workload significantly. Self-management functionalities such as self-healing and self-optimization could provide effective assistances for reduce down-time and increase system reliability.

In order to apply self-management, the autonomic service management is essential. The objective of applying autonomic service management is to enable distributed automation systems to manage themselves so that the human intervention could be minimized. The intention of the autonomic service management is to automate management function and externalize this function according to the behaviour defined by management interface [10]. In order to achieve a self-adaptive system, there must be an automatic method to collect all information required from the managed system. Then the collected information could be further analysed to determine if any change is required to be made. The changes must be specified as a plan or a sequence of actions. Finally, the change plan or the sequence of actions must be executed to form an intelligent control loop.

This paper aims for providing a more intelligent service-oriented architecture design by using the autonomic service management. The rest of the paper is organized as follows: In the Section II, the application of SOA concept in the industrial automation domain as well as existing approaches of autonomic service management is reviewed. In section III, the introduction of autonomic service management and application with SOA-based distributed automation system is described. An IEC 61499 resource model design based on the autonomic service manager and SOA is provided in the following section. In the section V, several key features of autonomic service management such as self-configuration and self-optimisation are illustrated. In the section VI, the concept will be applied to a case study. Finally, the paper is concluded in the section VII and recommendations for future work are provided.

II. Related Works

The service-oriented architecture is initially introduced into the industrial automation domain by Jammes et. al. [5]. The idea of drive the intelligence of computing and communications down to the device level is illustrated by adopting network-connected devices with service-oriented architecture and Web Services standards.

Lastra et. al. [6] [7] investigated benefits of using Web Services and Semantic Web technologies for SOA based non-time-critical automation systems. Ontological knowledge base is applied to factory automation systems to achieve flexible control. Based on the SOA concept, recovery techniques for fault handling of automation systems can be achieved.

There are also some agent-based approaches for enabling intelligence in industrial automation. Strasser et. al. [8] discussed the autonomous application recovery in IEC 61499 systems. The proposed approach is based on the concept of self-organisation where devices are capable to identify faults and recover automatically. However, many additional features must be implemented in the IEC 61499 standard in order to achieve this idea practically.

Another agent-based approach for dynamic reconfiguration of real-time distributed automation systems based on IEC 61499 is proposed by Brennan et al. [9]. The system model and agents are encapsulated into IEC 61499 function blocks. In order to achieving dynamic reconfiguration, agents for coordination, mobility and cohort are designed.

Autonomic Service Management reference architecture is originally proposed by IBM [10] for the computer science domain [11] and recently has been applied to many domains such as Grid [12], Home Automation [13], and telecommunication [14]. Autonomic service management is tightly connected with Service-Oriented Architecture generally.

Mezni et. al. [15] proposed an autonomic registry-based SOA model by combining self-* properties and policy-based management. The Web Service standard WS-Policy is used and extended with extra information on service specific adaptation actions. Also the Universal Description, Discovery and Integration (UDDI) are extended in order to support policy-based self-management. The model is implemented using a multi-agent based architecture where an individual agent is designed for providing service registry, operating as service provider and service planner. Also agents are capable for monitoring Quality of Service.

A set of policies and principles is defined for SOA governance by Parejo et. al. [16]. A SOA governance model is presented based on the OASIS reference model for SOA. Two new features process and policy are added to the model so that self-configuration, self-optimization, self-healing and self-protection can be achieved both in design time and at runtime.

Autonomic SOA concept is proposed by Bhakti et. al. [17] in order to achieve dynamically organizing topologies of services and interactions between the services. An interaction model of Autonomic SOA is also presented in a computational engineering framework.

Alaya et. al. [18] also designs a context aware extensible autonomic framework for machine-to-machine networks. Autonomic service managers could communicate with each other based on knowledge models and reasoning rules. The idea is demonstrated on a smart meeting use case. The ontological knowledge base model as well as SWRL rules for querying the knowledge base are provided [19].

Finally, there are several existing research results on autonomic computing which could be useful. Calinescu [20] proposes a reconfigurable service-oriented architecture for autonomic computing. A combination of automated code generation, model-based and object-oriented development techniques is presented to ensure that the framework can be used to add autonomic capabilities to systems. The data-centre resource management is illustrated as the case study.

Overall, the application of autonomic service management with SOA is well-proven in the general-purpose computing and communication world. The self-organizing intelligent control is enabled by introducing autonomic service management into SOA-based architecture. This will improve flexibility and interoperability significantly to fulfil intelligent, efficient and reliable distributed automation systems.

III. Applying autonomic Service Management in SOA-based distributed automation systems

As described in the previous section, in order to apply autonomic service management in distributed automation systems, a flexible and interoperable execution environment is necessary. In the previous work [21], an IEC 61499 compliant PLC runtime (with IEC 61131-3 programing languages supported) – Function Block Service Runtime (FBSRT) based on service-oriented architecture is developed. Each function block is operated as a service and events passed between function blocks are considered as messages between services. As a result, services (function blocks) could be created, modified and deleted dynamically without affecting normal function block execution. However, the dynamic reconfiguration was limited by sending management commands to the FBSRT manually. In this paper, the runtime structure will be extended with intelligence introduced by autonomic service management.

A. Introduction on Autonomic Service Management Reference Architecture

The autonomic service management architecture is implemented as an autonomic service manager, which is referred as a "MAPE-K" approach usually [22]. The autonomic service manager "MAPE-K" internal architecture proposed by IBM is given in Fig. 1.

The "MAPE-K" refers to a set of functionalities: Monitor, Analyze, Plan and Execute with Knowledge source support. The Monitor function polls information from managed resources via sensors and manipulate collected data to create symptoms that need to be further investigated. Symptoms are analysed by the Analyze function to decide whether any change is required to be made to the managed system. The decision is influenced by a local knowledge base. A change request is generated and passed to the Plan function and an appropriate plan is selected from the knowledge source according to the plan. Finally, the change plan is performed by the Execute function. The procedure is carried out to the managed resource via Effector. The entire control loop is supported by the knowledge source. The knowledge source consists of both syntactic and semantic data like symptoms, change requests, change plans and other policies. The knowledge source can be easily extended to perform additional tasks, for example, recognizing a new symptom or create a new change plan.

There are four key features of self-management: self-configuration, self-healing, self-optimisation and self-protection [22]. Self-configuration refers to the system ability to dynamically reconfigure itself according to changing environmental. Self-healing aims for detects faults and recoveries from breakdown automatically. Self-optimisation is to optimize system performance. Self-protection provides prevention from external threats to ensure system security.

As described previously [4], the IEC 61499 standard can be used to implement SOA in a nature way. In order to achieve self-manageable PLCs, an autonomic service manager is placed in each IEC 61499 resource. Each function (Monitor, Analyze, Plan and Execute) in the autonomic service manager is implemented as a software service. Overall, the autonomic service manager provides a management service which composes from those services.

Those four key features of autonomic management can be applied in industrial automation domain as follows. Self-configuration refers to automatic deployment of IEC 61499 system configurations as opposed to manual deployment by developers: applications will be allocated to available hardware resources by autonomic service managers. Self-healing means that PLC has the ability to discover faults on any PLC within the network and relocate function blocks operating on the faulted PLC to another available PLC. Self-optimisation mostly works on performance – if one PLC workload is minimal and can be covered by another PLC, all assigned function blocks will be shifted to the other PLC. This PLC will turn into energy conservation mode. Finally, self-protection is mainly for protecting PLCs from unauthorized external access. Detailed implementation of these features will be provided in the section V.

B. Autonomic Service Manager Design for IEC 61499 Resource

In the previous section, the responsibility of the autonomic server manager for IEC 61499 systems was clarified. In the following part, the design of autonomic service manager for IEC 61499 will be discussed. The architecture is illustrated in Fig. 1.

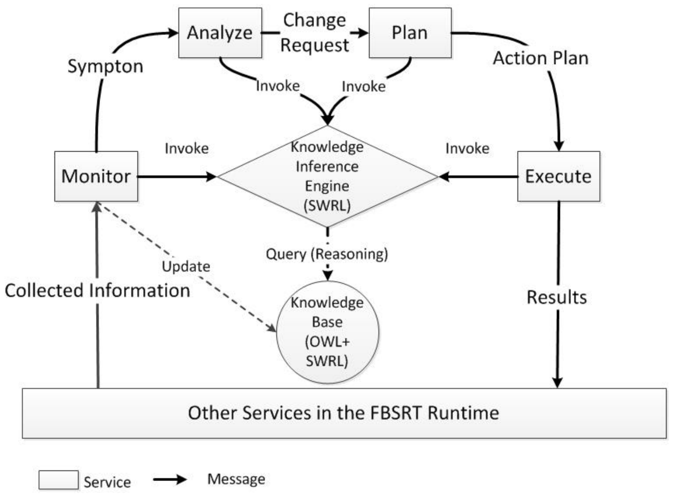

Fig. 1 Autonomic Service Manager Design for IEC 61499.

The managed resource refers to other services in the IEC 61499 runtime (details of those services are provided in the section IV). As stated previously, each function is also implemented as a software service. Information required by all functions is stored in a knowledge base using Ontology Web Language (OWL) [23] [25]. In the knowledge base, data are linked together by properties and presented in a logical way. A knowledge base inference engine is built to query knowledge base from all services. The inference engine is a rule-based query engine. Rules are defined in the Semantic Web Rule Language (SWRL) [24] that both human and machine can read and easily extend.

The knowledge base definition is given in the Fig. 2. There are four root objects in the knowledge base: Symptom, Change Request, Action Plan and Message. The monitor service subscribes to events from other services in the execution environment such as monitoring service, which provides information about hardware healthy status (power supply and battery percentage) and performance (last and worst execution time of last event). Each message contains information about source service, type of message data and its value.

Symptoms generated from the message data are filtered by the SWRL rules. For example, a symptom - *NotResponding* is discovered when a particular function block service is not responding to the monitoring service. The situation is described as the following SWRL rule in the Fig. 3.

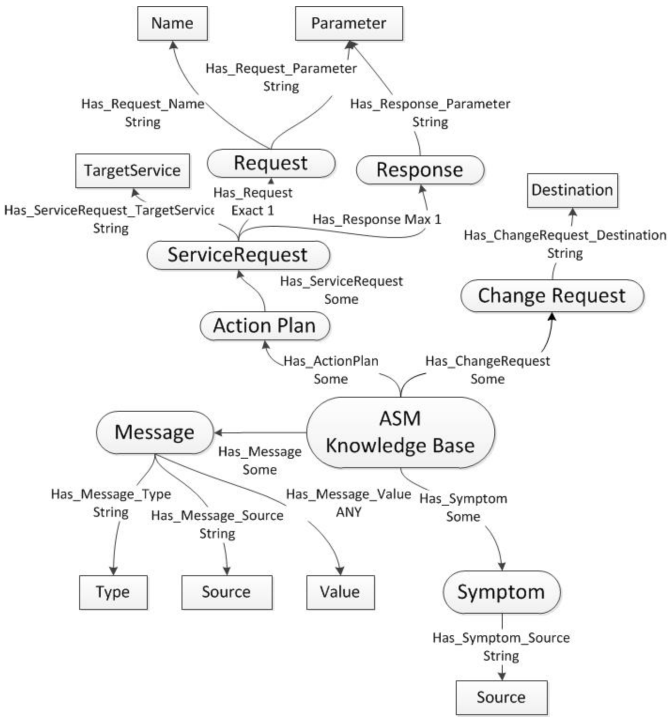

Fig. 2 Knowledge Base Definition for IEC 61499 Resource Autonomic Service Manager.

```
Message(?m) ^ Has_Message_Type(?m, ?type)
^ swrlb:stringEqualIgnoreCase(?type, “NotResponding”)
^ Has_Message_Source(?m, ?fbsource)
^ Has_Message_Value(?m, ?value) ^ swrlb:equal(?value, 0)
-> Symptom(NotResponding)
^ Has_Symptom_Source(NotResponding, ?fbsource)
```

Fig. 3. Example SWRL Rule for Symptom

The symptom will be further analysed by the Analyze service to identify what change are required to be made. When a service component is not responding to the monitoring service, a change request of restarting this function block will be raised. The following SWRL rule is created for creation of the change request.

```
Symptom(NotResponding)
^ Has_Symptom_Source(NotResponding, ?fbsource)
-> ChangeRequest(RestartFBService)
^ Has_ChangeRequest_Destination(RestartFBService, ?fbsource)
```

Fig. 4. Example SWRL Rule for Change Request

The last step in the ASM loop is to select a proper action plan to complete the change request. An action plan consists of a sequence of service requests. To invoke a function from a service, two things must be defined: function name and parameter values. The invoked function may also return a parameter back. The action plan example of a function block restart is given as stated in Fig. 5.

```
ChangeRequest(RestartFBService)
^ Has_ChangeRequest_Destination(RestartFBService, ?fbsource)
-> ActionPlan(RestartFB)
^ Has_ServiceRequest(RestartFB, ?service)
^ Has_ServiceRequest_TargetService(?service, ?fbsource)
^ Has_Request(?service, ?req)
^ Has_Request_Name(?req, ?name)
^ Has_Request_Parameter(?req, ?reqpara)
^ Has_Response(?service, ?res)
^ Has_Response_Paramter(?res, ?respara)
```

Fig. 5. Example SWRL Rule for Action Plan

The selected service sequence finally will be executed by the Execute function.

IV. SOA-BASED MODEL FOR IEC 61499 RESOURCE

In this section, a complete SOA-based model with autonomic service manager built-in for IEC 61499 resource will be provided. The SOA-based IEC 61499 resource model is illustrated in Fig. 6.

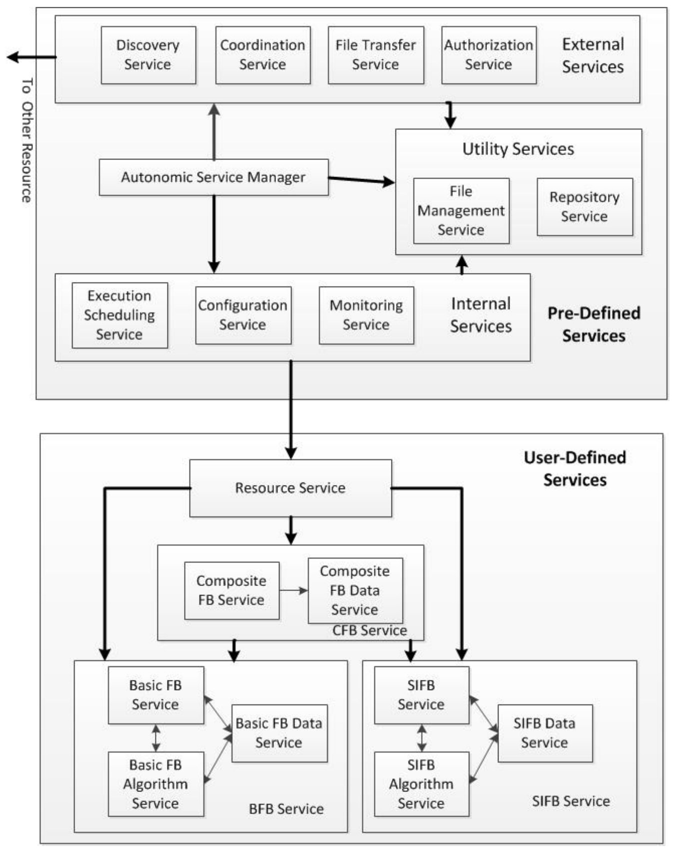

Fig. 6 SOA-Based IEC 61499 Resource Model.

There are two types of services in the model: pre-defined services and user-defined services. Pre-defined services are the built-in functionalities in the IEC 61499 resource model which cannot be deleted or modified. The autonomic service manager is the core service in the pre-defined services. The autonomic service manager acts as “brain” for the IEC 61499 resource model. It communicates with other resource models via external services.

External services include discovery service, coordination service, file transfer service and authorization service as shown in Fig. 7.

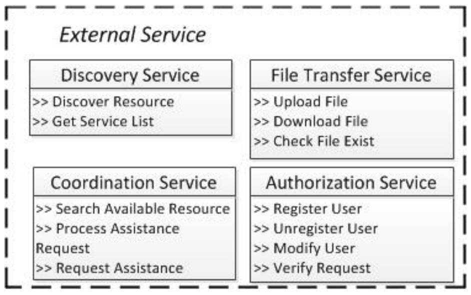

Fig. 7 External Service Definitions.

The discovery service is responsible for search other IEC 61499 resources on the same network. The Web Service discovery protocol – WS-Discovery is implemented in order to achieve discoverability [26]. Once another IEC 61499 resource is found by the discovery function, the address of the new IEC 61499 resource will be recorded by the Repository service. The coordination service provides cooperation mechanism with other IEC 61499 resources. The coordination service act as a communication gateway, which handles external assistance requests and negotiate with other available resources for assistance. The next service, the file transfer service, is mainly responsible for exchanging definitions of user-defined services between resources. New service definition will be placed to the correct path by the file management service. The last external service is the authorization service. The authorization service ensures there is no unauthorized connection can give orders to the autonomic service manager. A user could be registered, modified and unregistered from the Authorization Service.

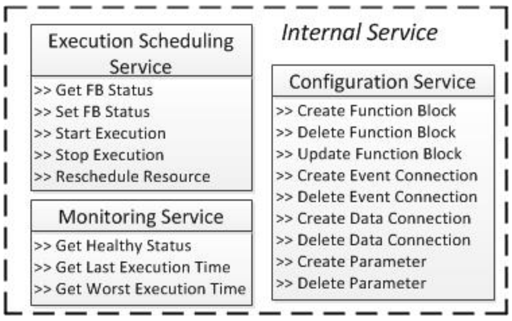

Fig. 8 Internal Service Definitions.

Secondly, internal services are responsible for controlling user-defined services as listed in Fig. 8. The configuration service is able to create or delete function blocks, connections and parameters dynamically during the operation. After function blocks are configured, they must be scheduled into the operation by the execution scheduling service. The execution scheduling service provides features such as modifying status of a particular function block, start/stop its execution and reschedule function block network in an IEC 61499 resource. Finally the monitoring service gathers status of each user-defined service. The data collected by the monitoring service will be used for assisting the autonomic service manager.

Implementation of IEC 61499 function block application is reflected as user-defined services. For each resource and function block type in the system configuration, a separate service definition is created. For each function block instance, an individual data service is added. The detailed design for basic, composite and service interface function block service are already discussed in the previous work [21]. The new service, resource service, is defined in Fig. 9. The resource service only contains static definitions of resource information and function block network encapsulated. An execution scheduler is designed for scheduling the execution order of function block network assigned in this resource. The execution scheduler could be configured by the Execution Scheduling Service.

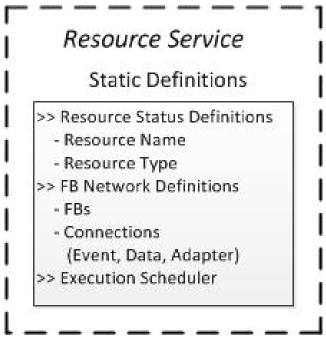

Fig. 9 Resource Service Definitions.

V. SELF-MANAGEMENT IMPLEMENTED VIA AUTONOMIC SERVICE MANAGER

To achieve efficient and reliable control of distributed automation systems, several self-management features will be demonstrated in this section. To provide better understanding, each feature will be illustrated using a service sequence diagram. The service sequence presented in each diagram is stored as an action plan in the knowledge base.

Firstly, self-configuration is demonstrated in Fig. 10. When autonomic service manager receives a request to deploy an IEC 61499 system configuration (programming tool will send the request to the first discovered autonomic service manager), it will determine if the local resource is capable to handle the entire system configuration. The get FB status and get worst-case execution time function are invoked by the monitoring service. The autonomic service manager can also send a "search other available resources" request to the coordination service. The coordination service utilizes the discovery service to identify whether any resources is available. The autonomic service manager then generates a deployment plan based on the capability of each PLC. If more than one resource is required, then an assistance request is sent by the coordination service. The local deployment plan is executed by the configuration services where all local function blocks, event and data connections and parameters are created. The final step is to reschedule the resource and start execution by invoking the execution scheduling service. The self-configuration provides the ability that the system configuration can be deployed without any human effort.

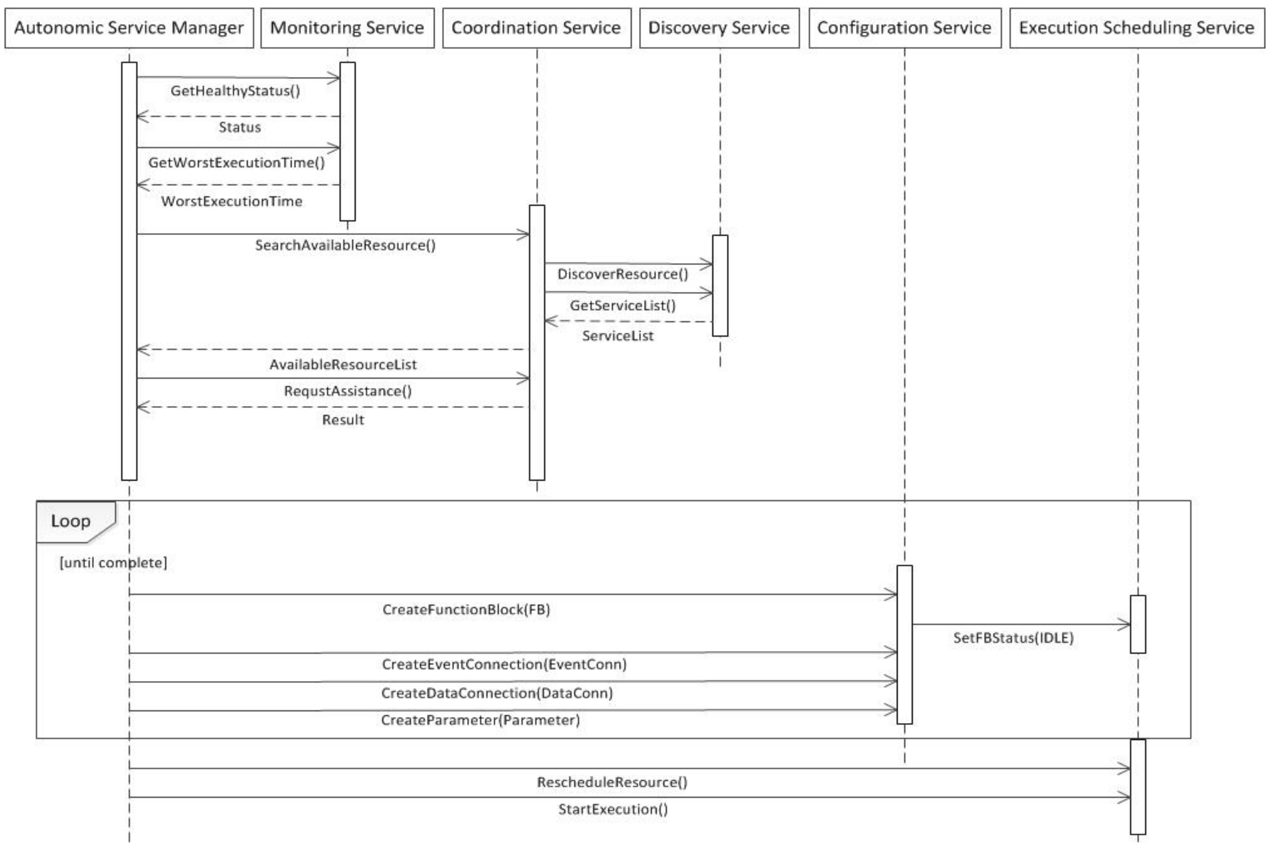

Fig. 10 Automatic Deployment (Self-Configuration) Service Sequence Definition.

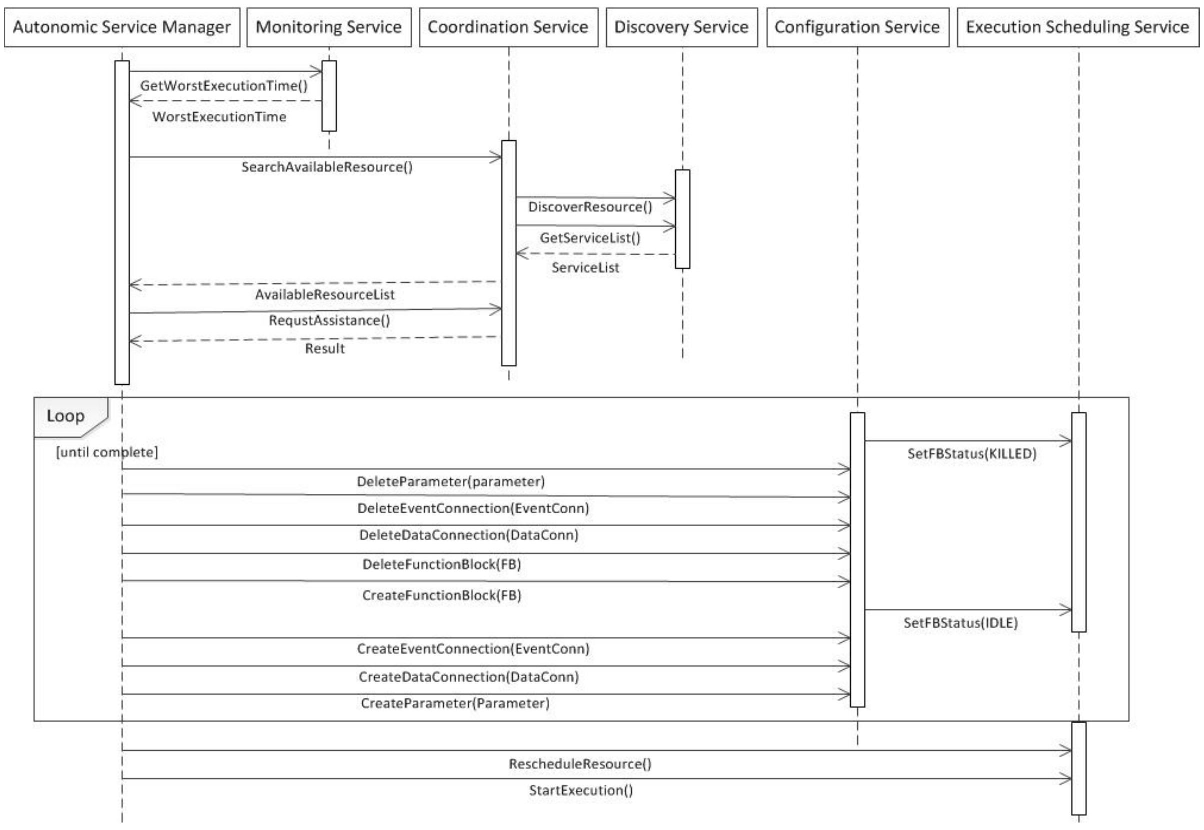

Fig. 11 Automatic Fault Detection and Recovery (Self-Healing) and Automatic Load Share (Self-Optimisation) Service

Both self-healing and self-optimization rely on the dynamic reconfiguration feature. When an autonomic service manager detects the resource which is not responding (self-healing) or the monitoring service issues an overload event when the worst execution time is longer than the I/O refresh rate (self-optimization), the autonomic service manager will start reconfiguration of the system as shown in Fig. 11. The procedure is similar to the self-configuration service sequence. For the self-healing case, when another resource is not responding, control logic assigned to that resource will be recreated on other available resources by the coordination service. In the case of self-optimization, an overload event will be generated when a resource is busy. Those function blocks will be created at the new allocated resource and the original definition will be deleted by the configuration service on the existing resource.

VI. Autonomic Service Management Case Study

A preliminary experiment on autonomic service management is performed on an airport baggage handling system subsystem. As shown in Fig. 12, the test case is a screening subsystem of the airport baggage handling system (BHS). There are four screening lines (CB-A, CB-B, CB-C and CB-D) which are feed by two induction lines (CI-A and CI-B). Bags are distributed to four screening lines by push diverters (PLD1 to PLD4). The bags cleared from the Exposure Detection Machines (EDS1 to EDS4) will be sent to the sortation system via CCx lines. Bags detected as suspicious will be sent to further investigation via CFx lines.

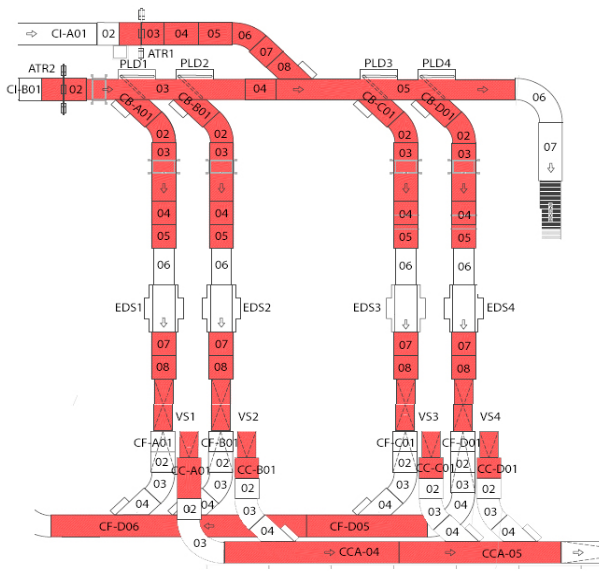

Fig. 12 Airport Baggage Handling System Layout.

During normal operation, large volume of bags is sent to manual inspection during peak hours. In order to reduce workload for operators, the queuing functionality – "indexing" intends to be added to every conveyor in the subsystem. The indexing functionality is enabled to accumulate bags when the downstream conveyors are stopped. Conveyor leaves indexing mode when the downstream line returns to full flow automatically.

In order to achieve that, the ECC in the conveyor control BFB (type *FB_Conveyor_I)* must be modified by inserting a new EC state "INDEX". The steps of the reconfiguration sequence are listed in Table 1 below which adds the queuing functionality without stop normal operation. The sequence for changing every conveyor control FB is: stop this FB instance; delete all connections from/to this FB instance; delete this FB instance; Create new conveyor control FB type; create new FB instance; Re-create all connections and restart this FB instance.

Steps	Management Commands
1: STOP CBA03 FB (Type:FB_Conveyor_S)	<Request ID="1" Action="STOP"> <FB Name="CBA03" Type="FB_Conveyor_S" /> </Request>
2: DELETE all Event and Data connections from/to CBA03 FB	<Request ID="2" Action="DELETE"> <Connection Source="…" Destination="…" /> </Request>
3: Delete CBA03 FB	<Request ID="3" Action="DELETE"> <FB Name="CBA03" Type="FB_Conveyor_S" /> </Request>
4: Create FB Type: FB_Conveyor_I	<Request ID="4" Action="CREATE"> <FBType Name="FB_Conveyor_I" > … // FB Type Definition for FB_Conveyor_I </FBType> </Request>
5: Create FB CBA03 (Type:FB_Conveyor_I)	<Request ID="5" Action="CREATE"> <FB Name="CBA03" Type="FB_Conveyor_I" /> </Request>
6: Create all Event and Data connections from/to CBA03 FB	<Request ID="6" Action="CREATE"> <Connection Source="…" Destination="…" /> </Request>
7: Create new data connection for Index	<Request ID="7" Action="CREATE"> <Connection Source="CBA03.Index" Destination="CBA02.PrevSend" /> </Request>
8: Start FB CBA03	<Request ID="8" Action="START"> <FB Name="CBA03" Type="FB_Conveyor_I" /> </Request>

Table. 1 Reconfiguration Sequence for Inserting Queuing Functionality.

VII. Conclusions And Future Work

A new IEC 61499 execution environment based on the SOA is proposed in this paper. The adoption of SOA at the device level improves flexibility and interoperability of industrial automation systems. The autonomic service manager is introduced as a software service in the runtime which provides intelligent self-manageable properties. Self-Optimisation improves the performance and the efficiency. Self-healing increases the reliability of the IEC 61499 system. The autonomic service manager is implemented using the Semantic Web technologies. Both OWL and SWRL are human and machine readable, more importantly knowledge can be easily extended. Rule-based query engine ensures new knowledge can be added seamlessly.

Continuing from this work, self-management features will be fully developed. Self-management rules will be formally defined. The performance of the autonomic service management will be analysed and the methodology of estimating and measuring worst case response time will be developed. Last but not the least, more complex use case will be used to demonstrate self-manageable distributed automation systems.

References

[1] IEC 61131-3, Programmable controllers - Part 3: Programming languages, *International Standard*, Second Edition, 2003

[2] IEC 61499, Function Blocks, *International Standard*, International Electrotechnical Commission, Geneva, Switzerland, Second Edition, 2012

[3] nxtControl GmbH, nxtStudio and nxtRT61499F - Next generation software for next generation customers [Online, 2009, June]. Available: http://www.nxtcontrol.com/

[4] W. Dai, V. Vyatkin and J. Christensen, "The Application of Service-Oriented Architectures in Distributed Automation Systems", *IEEE International Conference on Robotics and Automation,* accepted, 2014.

[5] F. Jammes and H. Smit, "Service-Oriented Paradigms in Industrial Automation," *IEEE Transactions on Industrial Informatics*, vol. 1, pp. 62 - 70, 2005.

[6] M. Uddin, A. Dvoryanchikova, A. Lobov and J. Lastra, "An ontology-based semantic foundation for flexible manufacturing systems", *IEEE Industrial Electronics Society Annual Conference,* Page 340 - 345, 2011.

[7] C. Popescu and J. Lastra, "Modeling breakdown handling for SOA-based factory automation systems", *IEEE International Conference on Industrial Technology*, Page 1 - 7, 2009.

[8] T. Strasser and R. Foschauer, "Autonomous Application Recovery in Distributed Intelligent Automation and Control Systems", *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, Vol. 42, Issue 6, Page 1054 – 1070, 2012.

[9] N. Cai, M. Gholami, L. Yang and R. Brennan, "Application-Oriented Intelligent Middleware for Distributed Sensing and Control", *IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews,* Vol. 42, No. 6, Page 947 – 956, 2012.

[10] IBM, "An Architecture Blueprint for Autonomic Computing (7th ed.)", white paper [Online, 2006], available from http://www-03.ibm.com/autonomic/pdfs/AC%20Blueprint%20White%20Paper%20V7.pdf

[11] M. Parashar and S. Hariri, "Autonomic Computing: An Overview", *Unconventional Programming Paradigms*, ISSN: 0302-9743, page 257 – 269, 2005.

[12] H. Liu, V. Bhat, M. Parashar and S. Klasky, "An autonomic service architecture for self-managing grid applications", *The 6th IEEE/ACM International Workshop on Grid Computing*, Page 132 – 139, 2005

[13] J. Bourcier, A. Diaconescu, P. Lalanda and J. McCann, "AutoHome: An Autonomic Management Framework for Pervasive Home Applications", *ACM Transactions on Autonomous and Adaptive Systems*, Vol. 6, Issue 1, Page 1 -9, 2011.

[14] C. Yu, A. Leon-Garcia and I. Foster, "Toward an Autonomic Service Management Framework: A Holistic Vision of SOA, AON and Autonomic Computing", *IEEE Communication Magazine*, Vol. 46, Issue. 5, Page 138 – 146, 2008

[15] H. Mezni, W. Chainbi and K. Ghedira, "An autonomic registry-based SOA model", *IEEE International Conference on Service-Oriented Computing and Application (SOCA)*, Page 1 -4, 2011.

[16] J. A. Parejo, P. Fern and A. Ruiz-cortes, "SOA Governance: Exploring Challenges & Benefits from an Autonomic Perspective", *International Conference on Software Engineering and Databases,* Vol. 3, No. 4, Page 52 – 61, 2009

[17] M. A. C. Bhakti and A. B. Abdullah, "Towards an autonomic Service Oriented Architecture in computational engineering framework", *10th International Conference on Information Sciences Signal Processing and Their Applications (ISSPA),* Page 741 – 744, 2010.

[18] M. B. Alaya, S. Matoussi, T. Monteil, and K. Drira, "Autonomic computing system for self-management of machine-to-machine networks", *Proceedings of the 2012 International Workshop on Self-aware internet of things,* Page 25 – 30, 2012.

[19] M. B. Alaya, T. Monteil and K. Drira, "Autonomic framework based on semantic models for self-management of ubiquitous systems", *Proceedings of the 2012 ACM conference on Ubiquitous Computing,* Page 860 – 862, 2012.

[20] R. Calinescu, "Reconfigurable Service-Oriented Architecture for Autonomic Computing", *International Journal on Advances in Intelligent Systems,* Vol. 2, No. 1, Page 38 – 57, 2009.

[21] W. Dai, V. Vyatkin, J. Christensen and V. Dubinin, "Bridging Service-Oriented Architecture and IEC 61499 for Flexibility and Dynamic Reconfigurability", *IEEE Transactions on Industrial Informatics*, submitted, 2014

[22] P. Lalanda, J. A. McCann and A. Diaconescu, "Autonomic Computing: Principles, Design and Implementation", *Springer,* 288 pages, 2013

[23] Ontology General Definition [Online], 2011, available from http://semanticweb.org/wiki/Ontology

[24] SWRL: A Semantic Web Rule Language Combining OWL and RuleML[Online], retrieved from http://www.w3g.org/Submission/SWRL/

[25] F. Badder, D. Calavanese, D.L. McGuinness, D. Nardi and P.F. Patel-Schneider, "The Description Logic Handbook, Theory, Implementation and Applications, 2nd Edition.", *Published by Cambridge University Press*, 2007, ISBN 978-0-521-87265-4

[26] WS-Discovery – Web Services Dynamic Discovery [Online 2009], Available: http://docs.oasis-open.org/ws-dd/discovery/1.1/wsdd-discovery-1.1-spec.html