I. Troch, F. Breitenecker, eds.   ISBN 978-3-901608-35-3

A Decentralized Foundation Model for Automation Technologies

H. Mersch[1], C. Kleegrewe[2], U. Epple[1]
[1]RWTH Aachen University, Germany,
[2]Corporate Technology Intelligent Autonomous Systems, Siemens AG, Germany

Corresponding author: H. Mersch, Chair of Process Control Engineering, RWTH Aachen University
52064 Aachen, Turmstr. 46, Germany
h.mersch@plt.rwth-aachen.de

Abstract. For current automation systems a device failure yields to extensive costs and working time for bringing a replacement in production mode. Beside the physical replacement itself the setup procedure of the newly added device produces workload for several engineers before the device is in the position to fulfill the demand of the failed device – most likely a complete workflow needs to be established.

This paper presents an foundation approach addressing this and other current issues within the field of automation. The approach bases on ideas from second generation Peer2Peer systems [6] focusing decentralization. The foundation provides an interface, which abstracts from the Peer2Peer overlay network and informs about network changes as well as provides a query mechanism for states of other peers. Based on this foundation and by using this interface applications in the field of automation could be realized, which are decentralized and capable of supporting maintenance as well as normal production procedure. It is proposed that these abstract from their implementation and thus build a service-oriented architecture (SOA) [4] on top of the presented model providing higher-level functionality.

A prototype realization basing on the ACPLT Technologies as described in [1] was implemented for validating the proposed approach. This demonstrates how configuration data of devices could be stored decentralizes & relyable and how this could be applied to the described failure-replacement-workflow through the introduced *Zero-Config* approach.

1 Introduction

The current automation systems are designed centrally. Central systems in general are easier to implement, realize and maintain. Nevertheless they have disadvantages like their central instance provides a single point of failure as discussed in e.g. [6]. The developed and presented model describes a new way of automation system design. This model bases on the idea of decentralized Peer2Peer systems of the information technology.

Peer2Peer. Peer2Peer systems base on an existing physical network - most likely a TCP/IP network. This layer is called *Underlay Network*, which means that the technology of the *Overlay Network* (see Figure 1), the Peer2Peer system has to be adjusted to capabilities of the Underlay Network. Where some second generation Peer2Peer networks have a central point within their overlay network – e.g. for authentification – this approach is completely decentralized thus a resulting system will be more complex in terms of design but even more flexible and reliable. Nevertheless centrally designed application design would still be possible to realize and might be applicable depending on application's focus.

SOA. Additionally this approach introduces the service-oriented architecture principle (SOA) for these kind of systems, thus proposes to see an overlay Peer2Peer network as an foundation for a wide range of decentralized applications consist of services, which could depend on each other as well as interact - illustrated in Figure 2. The approach proposes an abstract architecture model including the application interface as one main part providing information about the current Peer2Peer network and its changes to the high-level applications on top of the foundation. By abstracting from the concrete implementation as well as the management of data within the foundation this interface makes it possible to customize the approach to various application scenarios.

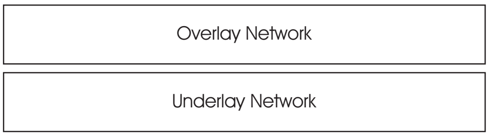

Figure 1: Classical Peer2Peer architecture – Upon an physical network (e.g. TCP/IP based) an overlay Peer2Peer network is build and abstracts from the architecture of the underlay network.

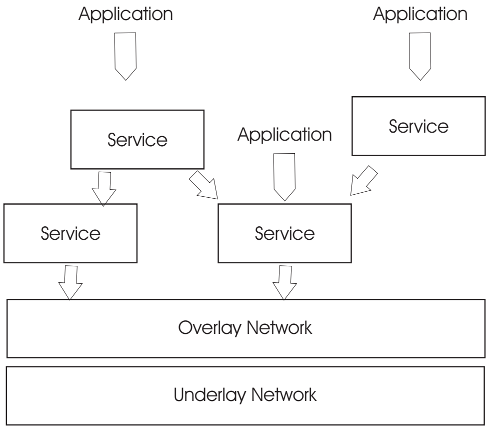

Figure 2: Proposed Peer2Peer architecture supported by this foundation model – Based on the foundation, which build the upper interface of the overlay network, SOA-based applications are build.

For realization one needs to define so called *flavors* for this foundation, which concretize the model by defining specific *Plug Points*. They need to fit together in a way that the whole system fulfills the concrete requirements. As an example one Plug Point is *Neighborhood* describing a set of peers, which are monitored by one peer – meaning through this peer the network recognizes and thus could react on a failure of the monitored devices. Different flavors would have different performance advantages as well as penalties. Like in all such complex systems the developer has the choice between speed (meaning less communication for Peer2Peer networks) and memory (storing less P2P network information) by defining an appropriate flavor.

Within the prototype of the foundation model, a simple flavor was designed and realized. For demonstrating the capabilities of this approach, this flavor assumes as plug point for the neighborhood the system to be rather small thus all peers could store information about the whole network. Other flavors address larger networks or devices not providing enough memory to store a whole view of the network. These flavors would – by demand of their upper layer services – retrieve information about the base network on the fly.

As an example of this foundation model an implementation of the *Zero-Config* scenario (detailed description in Section 3.2 is presented, which addresses the failure-replacement-workflow mentioned at the beginning. By storing the configuration data within the application *Distributed Data* as well as monitoring failures, the application *Self-Config* could upon startup of the device retrieve information about past failures and (potentially) decide which replacement the device is for. After acknowledge of the operator, the device could retrieve the configuration data and directly go to production mode.

2 Foundation – Concepts and Components

For presenting the foundation in detail the concept is briefly described first. This is followed by detailed descriptions of the different components and variations the foundation model provides.

As stated in the Section 1, the foundation itself is located within the Overlay Network (see Figure 2) and provides an *Application Interface* for applications, which base on the foundation and could interact between each other as proposed by the service-oriented architecture (SOA) principle [4]. This *Application Interface* provides information about the Peer2Peer network itself as well as notification upon changes of it. This leads to the idea, that the foundation of one peer observes other peers in a decentralized way. Those are called neighbors. By defining an identity number per peer, the foundation orders peers, which enables it to define which peer has which other peer as neighbor. This identity number bundled with a communication address (like IP Address for IP based communication) is called *Peer Descriptor*. Other tasks of the foundation are providing a JOIN procedure, thus a new peer could participate within an existing network, as well as holding information about the state of other peers - either partly or complete of the network as described later.

Definition 1 (Peer Descriptor) *The foundation assumes that every peer has a unique identification number, which builds the Peer Descriptor together with the corresponding communication address.*

Communication is an important part of a Peer2Peer system. Nevertheless, this foundation – as every Peer2Peer system – assumes a underlay network which enables devices to address other participants as well as enable communication. The foundation itself communicates (except one use case) via bidirectional point to point communication, which is supposed to be provided by each known network principle. If additionally a broadcast communication is provided (like UDP for ethernet), this could be used for discovering a *First Entry Point* (see section 2.3). Using the chosen communication system and communication ways, different messages are defined by foundation's com-

Incoming messages	Component	Outgoing messages
PROMOTE LEFT	NetView	
ALIVE	Monitor	LEFT
	Announcer	ALIVE
FWDLEFT FWDPROMOTE	Body	PROMOTE INIT ENDINIT
INIT ENDINIT	Recorder	FWDLEFT FWDPROMOTE

Figure 3: Overview of in-/outgoing messages with associated components of the foundation.

ponents to fullfill its demands. As an overview, Figure 3 shows all components of the foundation including their associated messages, which are sent or retrieved.

The later described prototype uses the ACPLT/KS communication system (see Section 3) realized over ethernet and TCP/IP.

As the foundation model could be used in various scenarios providing different resources and requiring different capabilities, the foundation model introduces the term *flavor* as a collection of concrete definitions building a relyable foundation for an application scenario.

Definition 2 (Flavor) *A flavor is a concrete characteristic implementation of the foundation model. It defines the behavior and data management for fulfilling the demands in between the underlay network and the application interface.*

This means that the neighborhood definition could vary (e.g. minimizing communication within the foundation) as well as the stored data within the NetView component (e.g. minimizing the required memory). All these customization parts of the foundation model are called *Plug Point*s. They need to be chosen carefully and need to fit together. Within the next sections a detailed description of all foundation model components is given. While the description is held general a concrete flavor is presented additionally, which is implemented by the prototype. Thus the reader is capable of understanding the prototype while having the complete overview of the foundation model. A summarizing list of the flavor's requirements and all plug points is described in Section 2.5.

Definition 3 (Plug Point) *A flavor must take several Plug points into account, which have to fit together for building a relyable and robust foundation.*

For an overview a list of the foundation components is given before detailed descriptions are provided:

- *NetView* Holding information about the current network change (thus other peers)
- *Monitor* Taking care of *neighbors* by monitoring received ALIVE messages
- *Announcer* Sending ALIVE Messages to neighbor's Monitor component
- *Body* Executing the Startup Procedure, thus discovery, identity decision and joining
- *Recorder* Providing remote information about NetView changes - used at startup of other peers
- *ApplicationContainer* Container of all applications using the foundation

2.1 Current state of Network: NetView

Applications as well as the foundation system itself are interested in the current state of the network, which is stored within the *NetView* component. This is basically a table holding information about known other peers as well as their state. Updates to this component could come from different information sources. On the one hand the *Life Management* (see next section) as well as the *Join Procedure* are explicit components for publishing and updating information about the current network state. On the other hand, feedback from the applications might be taken into account. Different flavors could store this view partly or complete – updated online or on request, as described in more detail in Section 2.5.

For the prototype a complete view of the network, thus all information about other peers, is stored locally at each peer. Feedback of the applications to the foundation isn't supported since demonstration purposes would be even harder to present in a clear and understandable way.

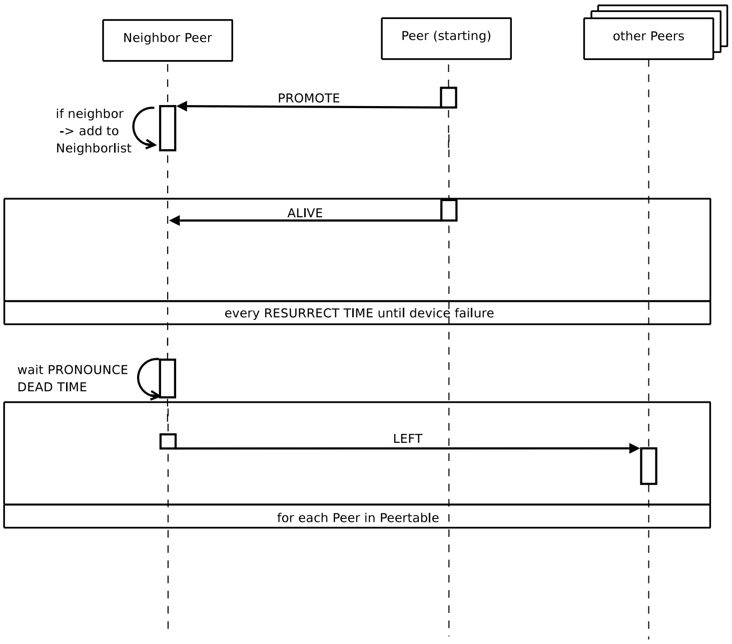

Figure 4: Neighborship initialization and monitoring – After establishing the neighborhood connection ALIVE messages are sent regularly. If these aren't received by the neighbor, LEFT messages mark this failed peer as OFFLINE.

2.2 Life Management: Monitor / Announcer

As described at the beginning of this section, the foundation approach suggests to observe other peers, which are called *neighbors*.

Definition 4 (Neighbor) *Neighbors are observing each other: They send regularly ALIVE messages to their neighbors indicating they exist. If no ALIVE message is retrieved for a time interval defined by the Pronounce Dead Time, the peer is marked to be offline network-wide.*
Neighborhoods are defined to be bijective, thus if a peer A has peer B as neighbor, peer B has also peer A as neighbor.
The neighborhood definition describes which peers are neighbors to each other.

The *Monitor* component retrieves ALIVE messages and stores the time of last received message per neighbor. These are sent out by the *Announcer* component of the neighbors regularly as defined by the network wide unique *Resurrect Time*. If a monitor does not retrieve an ALIVE message as long as the *Pronounce Dead Time* defines, LEFT messages are sent out to mark the peer OFFLINE within the Peer2Peer network. The procedure is presented graphically in Figure 4.

Definition 5 (Resurrect Time) *Network wide unique time interval between two ALIVE messages sent from the Announcer to neighbors' Monitor.*

Definition 6 (Pronounce Dead Time) *Network wide unique time interval defining time until a neighbor is marked network-wide offline , if no ALIVE message was retrieved. This should be a multiple of the Resurrect Time.*

For our prototype implemention's flavor a two-time neighborhood definition was chosen:

Definition 7 (Neighborhood definition for prototype implementation) *For Peer with ID* `X` *yields*
Peer `lN` *is lower neighbor, iff* `lN` *is peer with largest ID and ID < `X`, which is ONLINE*
and
Peer `uN` *is upper neighbor, iff* `uN` *is peer with smallest ID and ID > `X`, which is ONLINE.*
`lN` *and* `uN` *build the neighborhood of* `X`

This yields to a robust system, which is also easy to understand as well as to analyze and demonstrate as illustrated in Figure 5.

2.3 Startup Procedure: Recorder / Body

Once a device is started, the peer software will initialize itself. As described in Section 2 starting a peer requires two information items before joining an existing network:

1. Own identity (own peer descriptor) - underlay network address and network wide peer ID

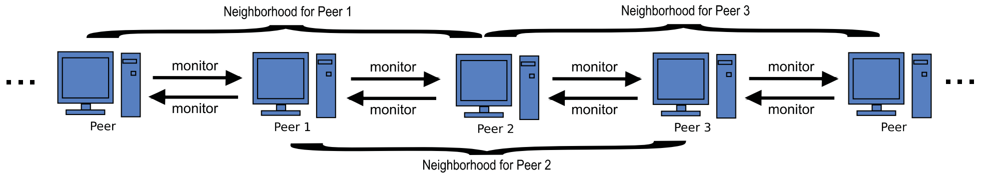

Figure 5: Neighborhood illustration for prototype flavor – A peer's neighborhood is the next higher and previous lower peer ID, which are online.

2. First Entry Point - an other running peer for retrieving network information and processing the startup

Thus the procedure could be splitted into three parts, whereby the first two are optional depending on provided information by the starting authority[1]. Both the FEP and the own identity could be provided as startup parameters, thus these steps could be skipped.

Definition 8 (First Entry Point (FEP)) *For joining an existing network a starting peer needs one initial contact to the network. This isn't required to be a later neighbor but any peer of the network, which is in normal operating mode.*

1. Discovery. First the initial way of starting communication with the network is defined via a FEP (First Entry Point), which needs to be an running Peer. This means, if the peer is started without knowing a *First Entry Point*, it requires to search for one. For discovering its FEP a peer needs to take capabilities of the communication system (the underlay network as introduced in Section 1) into account. Thus discovery could be distinguished into two scenarios:

- *Broadcasting provided* by underlay network means a broadcast message could be sent out and one/many FEP candidates could answer.
- *Broadcasting not provided* by underlay network means the peer needs to iterate over the address space until a FEP candidate is reached.

This is proposed to be realized by calling peer's applications via a specific message type, thus even this mechanisms could be plugged and customized.

The prototype is limited to a TCP/IP communication by its communication system meaning it iterates over the own subnetwork for finding a appropriate FEP candidate.

2. Identity decision. If no own identity is provided on startup, the foundation requires to either generate one carefully checking upon collisions while joining the network or retrieving one depending on state information by other peers. For decoupling this part, the foundation calls their applications using a special message type like proposed for the discovery part. This means that special applications – like the later described *Zero-Config* (see section 3.2) – could define a identity of the starting peer, which is provided to the foundation.

3. Join. Last part of startup procedure is the Join process itself. It is required that the starting peer retrieves information about the current network state, thus the information for its NetView component – this is handled by the Body component of a peer. For using an existing peer as a FEP a INIT message is transfered to the *Recorder* component of the FEP. The *Recorder* component provides information about the data stored at the local NetView by sending a INITRES message as answer. Further on it will inform the starting peer about changes (FWDLEFT and FWDPROMOTE messages) of other peers until an ENDINIT message is sent to the recorder. This is necessary since the starting peer might need to register within the network at other peers. During this time frame, the network might change and (since the new peer is not yet completely joined) it is not appropriate informed as defined by the flavor definition, which could lead to an inconsistent view to the P2P network.

For the prototype flavor – as the whole NetView table is stored at each peer – transferring the NetView information from the FEP to the new peer is appropriate. Afterwards each peer needs to be informed about existence of the new peer by sending a PROMOTE message. Figure 6 illustrates this Join procedure for the prototype implementation.

2.4 Using the foundation: Application Interface

As motivated at the beginning of Section 2, the application interface, which is provided by the foundation to all applications is the most important component.

Definition 9 (Application Interface) *Application interface is the interface between the overlay network and the (service-oriented) applications, which base on this foundation. It's aim is to hide as much details from the decentralized system while offering information thus applications could react on changes of the overlay network.*

[1] The application *Zero-Config* in section 3.2 does use both procedures

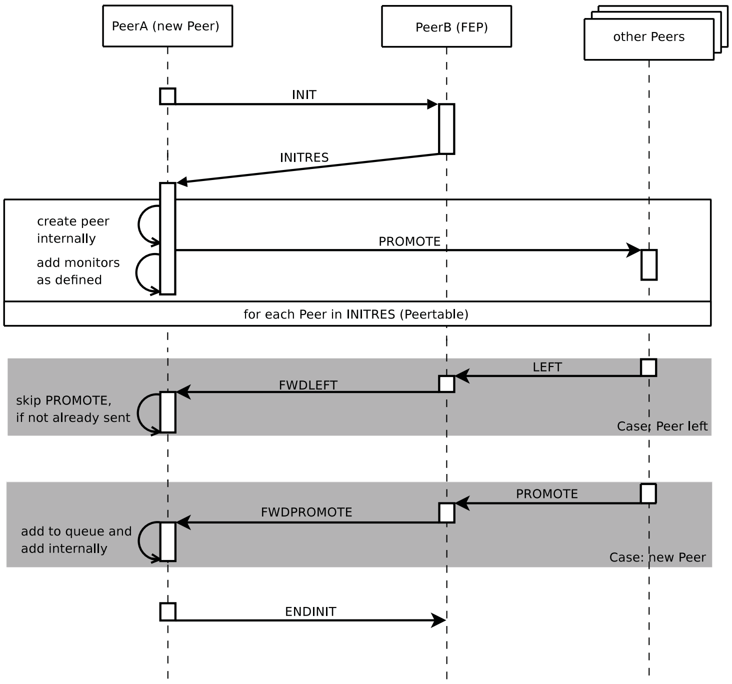

Figure 6: Join procedure of a starting peer with its (already discovered) FEP – New peer sends out an INIT message to the First Entry Point (FEP), which is answered by an INITRES message. This contains (maybe partly, depending on flavor) information about other peer. These are contacted by new peer using PROMOTE messages. If network changes during sending PROMOTE messages, these changes are forwarded by FEP peer until the new peer sends a ENDINIT message.

High-level services, which could abstract from the distribution and decentralization aspect providing an interface on application semantics will be provided by applications, which base on this application interface, as the SOA principle suggests.

The foundation provides two way communication via this interface.
First a **pull mechanism** provides application upon request with state information about a given peer ID, its descriptor or convenient methods like retrieving information about next higher ID which is online/offline. This is a query interface to the NetView component, which stores (or could retrieve as defined by flavor) information about any peer within the network.
The second communication way of the application interface is a **subscription & notification mechanism**. After registering by the application at the interface, each change of peers the application is interested in will be pushed forward to the applications thus they are in the position to react upon network changes.

For detecting changes of the overlay network more frequent than waiting for Pronounce Dead Time as described in Section 2.1 an application might give **feedback** about other peers as well. If an application couldn't contact the destination device for communication it might be clever to check the state of this peer even it isn't defined to be an neighbor by the neighborhood definition. This is a speeding up method, which complicates the overall network without providing new functionality and is not investigated any further until now. For the same reason this is not implemented for the prototype as it also would eliminate demonstration purposes since a failed peer might be detected on various ways.

Note. Even Life Management could be interpreted as application - they are registered at this application interface for getting notified, if the NetView information changes and neighborhood might need to be updated. This yields to the foundation capability that non-monitoring is also realizeable, nevertheless the foundation will loose its capability of recognizing failures and any changes.

2.5 Adjusting the foundation: Different Flavors

The foundation model addresses different application areas as well as different underlay networks – both might differ within their requirements and demands. This yields to the presented approach, which is customizeable for fullfilling the demands of various application areas like described in Section 1. For implementation of the foundation these customization *Plug Point*s need to be chosen carefully, thus the foundation provides relyable and powerful (in term of the application area) stability as well as information.

While the different components along with their variations (*Plug Point*s) are described in detail within their associated components above, just a brief summary is given here:

- *Neighborhood* Used by monitor, the neighborhood defines which other peers have to be monitored
- *Startup procedure* How to discover a First Entry Point, choose identity and receive (partly) NetView data.
- *Monitoring* Upon left peers, which peers need to be informed
- *NetView storage* Which network information is stored at a peer and how is non-stored information retrieved. This affects the Startup Procedure and Monitoring since PROMOTE/LEFT messages need to be sent out at least to peers, which have to take the starting peer into account. Minimum will always be to publish the neighbors as defined by the neighborhood definition.

3 Prototype implementation and application's scenario

For proving the presented concept a prototype implementation was developed. The prototype implementation realizes the foundation using the above described flavor. Thus it is assumed that the peer system runs on devices, which are capable of storing information about up to 255 peers. Furthermore the prototype's discovery mechanism is limited to search for FEPs within the own sub-network. This yields to a flexible peer setup for a rather small peer network, which builds the focus for the prototype implementation.

The prototype is realized using the ACPLT Technologies, which build the environment for the peer components. The ACPLT Technologies [2] are developed by the Chair for Process Control Engineering [5] at the RWTH Aachen University. They offer a runtime environment for object oriented model approaches not specifically for process control, but for any kind of object oriented implementations. Two established basic components of the ACPLT Technologies are ACPLT/KS and ACPLT/OV. ACPLT/KS is a communication protocol and implementation. This defines a basic set of actions on behave of this RPC [7] standard, thus defining communication for the different ACPLT Technology components. ACPLT/OV uses ACPLT/KS for communication and maps services and the model of KS to it's object oriented environment. This environment is compareable to modern object oriented programming languages, but is implemented in ANSI-C, which offers the possibility to run on micro controllers and other kind of small scale systems. Most of the principles of object oriented programming languages are represented in ACPLT/OV while others differ e.g. for enabling explorability: Any client could, via ACPLT/KS, communicate with an ACPLT/OV server and explore existing objects, values and associations (relations).

Every prior described foundation component as well as the applications are one or more ACPLT/OV objects in this term. As the ACPLT environment and the peer components as well base on ANSI-C code, any communication to other applications (like traditional automation software solutions) could be implemented. For a further demonstration setup, the later described *Zero-Config* application will configure external software showing the interaction capabilities between the prototype and existing process control systems.

The following sections describe two applications, which were implemented on this foundation prototype. Please note that these two descriptions do not handle all details of their design as well as their implementation since the focus of this paper is on the foundation system itself.

3.1 First application: Self-healing Redundant Data

For illustrating the usage of the application interface, the *Self-healing Redundant Data* application was developed and is described here. This application does rely on the application interface and stores a given key and value data pair redundantly within a *Replication Group*.

Definition 10 (Replication Group) *Group of online peers, which hold one data set composed of a key and a value. Members of a Replication Group are defined per key by the replication group definition (an hash function).*

For achieving this goal, a distributed hash table algorithm, which is network wide unique, is used. This defines deterministically the replication group for a given key - a group of online peers, storing the key/value pair (see application interface's pull mechanism). As a replication group depends on the peers, which are online, upon retrieving of LEFT or PROMOTE messages (see application interface's subscription & notification mechanism), the application will reorganize data sets, if replication group is affected. By always trying to reach *Replication Factor*, this application is self-healing as each data set will be stored *Replication Factor* times as long as there are enough peers online to reach this goal.

Definition 11 (Replication Factor) *Size of the Replication Group.*

Definition 12 (Replication Group definition of prototype) *The replication for a given key starts with the peer ID of the length of the key or next larger peer ID, which is online. Additionally the next larger peer IDs, which are online, are added until Replication Factor is reached.*

The application is used by the Zero-Config application, which is presented next.

3.2 Application scenario: Device failure handling with *Zero-Config*

Within Section 1 a problem was raised, which came from a device failure within an automation system and its reaction for current automation systems was described. This example application scenario targets to reduce this amount of workload while a device needs to be replaced.

For each device configuration data is stored using the prior described *Self-healing Redundant Data* application, where the ID of the device/peer is used as the key. Additionally, this application stores failures, which are recognized as LEFT messages coming from the foundation to all applications (see application interface's subscription & notification mechanism). If a device fails or needs to be replaced for any reason, the newly installed and starting device could retrieve this list of failed devices – based on the prior described discovery / identity decision mechanism (see Section 2.2). Afterwards a check might be performed for comparing own capabilities with the failed devices using the configuration data. This yields to a decision which failed device could be replaced by the new one and (probably after a confirmation with the operator) the device could setup itself using the configuration data, which is referenceable using the Self-healing Redundant Data application. Afterwards starting of any application (like device specific programs) and go into production mode is possible. The new device fullfills the same role within the automation system as the replaced one.

If appropriate resources exist, this approach would be possible to integrate into existing automation solutions – even on level of field devices, since configuration data might be stored and communication with the automation system (like notification of the new device) could be done as one specific application based on Zero-Config without interfering with the existing system.

This second application uses the first *Self-healing Redundant Data* for storing/retrieving configuration data. This shows the benefits of the SOA approach: The Zero-Config application does not need to care about how the specific implementation of the used services (applications) work – it uses and relies on correct and relyable design of the *Self-healing Redundant Data* as well as the foundation itself.

4 Conclusion and Outlook

The presented model yields to a new way of device communication within the automation technologies. The foundation as part of the overlay network (as defined by Peer2Peer terminology) provides an interface, which abstracts from the detailed implementation of the decentralized system. It's task is to provide information about the network as which peers are in shape and notify, if any peer changes its state within the network. The model suggests that peers monitor each other for recognizing failures as well as providing mechanisms to process network change requests like a starting peer joining the network. On top of this foundation a service-oriented architecture was proposed. This means that various applications, which use the foundation should use each other without taking care of the detailed implementation of the other applications. This was demonstrated with the presented *Zero-Config* application using the *Seal-healing Redundant Data* application without knowing how the stored configuration data is managed.

The *Zero-Config* application presents the benefits of the system: It enables a replacement of an failed device without configuration or setup maintenance - after discovering a failed device, which fulfills the same functionalities like a new one, the configuration data could be retrieved and appropriate steps are processed thus the replacement device could go into production mode as the failed was before.

The approach enables a complete decentralization by providing a migration path due to the fact that – as shown within the prototype – configuration could be implemented using the approach without interfacing the traditional system, which could be in operation like before. Other scenarios as heterogeneous peers [3] including discovery capabilities are planed to be adopted to this approach and would enable the foundation to handle centralized applications, which was – depending on scenario – proposed in Section 1 for specific tasks.

Currently, the prototype implementation was developed for Linux and Windows standard and embedded systems. A demonstration system based on industry process control hardware is planed to be realized as next step.

5 References

[1] H. Albrecht, *On Meta-Modeling for Communication in Operational Process Control Engineering*. VDI Fortschrittsbericht Reihe 8 Nr. 975 ISBN: 3-18-394008-6, 2003.

[2] *ACPLT Technologies Overview*, http://www.acplt.org, 12/2008.

[3] D. Drinjakovic, U. Epple, *Search methods in P2P-networks of process control systems*. 2nd IEEE International Conference on Industrial Informatics (INDIN'04), IEEE Catalog 04EX863, S. 101–107, 2004.

[4] T. Perl, *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*. Prentice Hall PT, ISBN 978-0131858589, 2005.

[5] *Chair of Process Control Engineering*, http://www.plt.rwth-aachen.de, 12/2008

[6] R. Steinmetz, K. Wehrle (Eds), *Peer-to-Peer Systems and Applications*. Lecture Notes in Computer Science Volume 3485, ISBN 3-540-29192-X, 2005.

[7] Sun Microsystems, Inc., *RPC: Remote Procedure Call Protocol Specification Version 2*. RFC 1057, 1988.