Integrating Recovery Strategies into a Primary Substation Automation System

G. Deconinck\textsuperscript{1}, V. De Florio\textsuperscript{1}, R. Belmans\textsuperscript{1}, G. Dondossola\textsuperscript{2}, J. Szanto\textsuperscript{2}
\textsuperscript{1}K.U.Leuven-ESAT, Kasteelpark Arenberg 10, B-3001 Leuven, Belgium
\textsuperscript{2}CESI, Via R. Rubattino 54, I-20134 Milan, Italy
\{gdec, deflorio, robbie@esat.kuleuven.ac.be\}, \{dondossola, szanto@cesi.it\}

Abstract

The DepAuDE architecture provides middleware to integrate fault tolerance support into distributed embedded automation applications. It allows error recovery to be expressed in terms of recovery strategies, i.e., lightweight code fragments separated from the application code. At run time, the middleware orchestrates their execution. This paper reports on the integration of different recovery scripts into a distributed run-time environment applied to the embedded automation system of a primary substation. An instrumented automata-based design environment allows the application to be deployed on a heterogeneous platform with several real-time operating systems. While the middleware detects the errors and selects the correct recovery scripts to be executed, the application functionality is maintained through system reconfiguration or graceful degradation. The added value comes from the flexibility to modify recovery strategies without requiring major modifications to the application, while tolerating the same physical faults as in the dedicated hardware solutions.

1 Introduction

Industrial distributed embedded systems –like those used in the control and automation of electrical energy infrastructures– rely on off-the-shelf components and protocols to ensure cost-efficient exploitation [3], [9]. As a particular application can be deployed on a variety of hardware targets (with different sets of sensors and actuators attached) and within different environments (e.g. with different levels of electromagnetic interference (EMI)), flexibility is needed both to instantiate the application functions appropriately and to react adequately to disturbances to the information and communication infrastructure on which the application is running. For instance, system reconfiguration and recovery may be different, depending on which I/O devices are connected to the different parts of the distributed controllers. More generally, adaptability is required to modify fault tolerance strategies depending on the environment.

The DepAuDE architecture deploys a set of middleware modules to provide fault tolerance by exploiting the embedded systems’ distributed hardware and by separating functional behavior from the recovery strategy, i.e., the set of actions to be executed when an error is detected [4], [5].

This architecture has been integrated in an innovative demonstrator of a Primary Substation Automation System (PSAS), i.e. the embedded hardware and software in a substation for electricity distribution, connecting high voltage lines (HV) to medium voltage (MV) lines over transformers. The PSAS requires protection, control, monitoring and supervision capabilities. It is representative of many applications with dependability requirements in the energy field [10]. The major source of faults in the system is electromagnetic interference caused by the process itself (opening and closing of HV/MV switchgear) in spite of the attention paid to designing for electromagnetic compatibility. Software and hardware faults in the automation system have to be considered as well. These cause errors in communication, execution and memory subsystems.

In the ongoing renewal of electric infrastructures in Europe, utility companies are replacing their dedicated hardware-based fault tolerance solutions by commercial, interconnected platforms. This trend is motivated by the growing need for more functionality: development of new, dedicated (hardware-based) solutions is considered too expensive and not flexible enough to keep up with the evolving requirements of the liberalized electricity market. The required dependability is reached by exploiting hardware redundancy in the distributed platform, combined with software-implemented fault tolerance solutions at the middleware level. Although software-based fault tolerance may have less coverage than hardware-based solutions, this was not considered inhibitive, because the physical (electrical, non-programmable) protection in the plant continues to act, as a last resort, as a safeguard for non-covered faults. Besides, high-quality software engineering and extensive on-site testing remain important to avoid introduction of design faults that could hamper mission-critical services.

This paper presents the experience of integrating this DepAuDE software architecture into an application developed with the ASFA distributed run-time environment [11], which is a prototype of a new embedded control system for a primary substation. According to the approach proposed here, support for allocation of tasks to components, for reactions to detected
errors and for maintainability of the fault tolerance strategy is accomplished by using the configuration-and-recovery language ARIEL, a key component of the DepAuDE architecture. Section 2 describes the key elements of the DepAuDE approach and its underlying assumptions, while section 3 describes the PSAS application and the instantiation of the DepAuDE architecture therein. Section 4 concludes with a qualitative evaluation of the experience.

2 DepAuDE approach

A key component of the DepAuDE approach is ARIEL, a configuration-and-recovery language used to configure the different elements of the middleware architecture (e.g. allocate functions to tasks and nodes; set parameters) and to express recovery strategies - i.e., to describe diagnosis, containment and recovery actions to be executed when an error is detected [5], [8]. As such, it is possible to start a standby task, to reset a node or link, to move a task to another node, to generate synchronization signals for reconfiguration, etc. Ariel expresses error recovery in terms of IF guard THEN action [ELSE action] statements. The guard is a Boolean expression resulting in queries to the database of the BackBone. The THEN (ELSE) keyword marks the beginning of a list of fault tolerance actions to be executed when the guard is evaluated as true (false). Three sets of software modules build up the middleware architecture [4], [5].

• The Basic Services Layer (BSL), providing a real-time operating system (RTOS) interface for task handling and inter process communication; beside the library functions, it consists of two tasks: NodeController (to start & stop tasks) and SocketServer (to send & receive messages using sockets and mailboxes).

• A set of Fault Tolerance Mechanisms (FTM), performing error detection, network and task monitoring, fault masking, etc.

• The BackBone, i.e., a distributed application that maintains information on application progress as well as on the system topology and status. The BackBone receives event notifications generated by BSL, FTM and the user application, and retains this information in a database. When notified of a detected error, it interprets recovery strategies written in ARIEL by querying the database to assess the system status in order to orchestrate fault tolerance actions. The BackBone includes self-healing mechanisms.

2.1 Network and application model

An application runs on several nodes of a particular site, interconnected via an intra-site (local) network. This intra-site network is dedicated to the application, and it provides adequate real-time support. The focus of the case study of section 3 is on this intra-site architecture. The application can interact with applications on other sites over an inter-site (wide area) network, via gateway nodes. This inter-site network may be a non-dedicated, open network -such as the Internet- shared with other applications, and hence not under application control. A gateway node performs inter-site communication using tunneling and Redundant Source-Routing (multiple messages over disjoint paths) in order to increase inter-site connection availability and reduce message latency [12].

2.2 Assumptions

DepAuDE relies on the following assumptions.

• Fault model: a single physical fault affects execution or communication entities (tasks, nodes, links). Experiments confirm that EMI affects only the entity to which the responsible I/O element is connected [10]. Depending on the underlying hardware and RTOS (i.e. if a memory management unit is available), a fault containment region is a task or node. Crash failure semantics (fail-silent behavior) is assumed for the fault containment region.

• A synchronous system model is assumed (i.e. known & bounded processing delays, communication delays, clock differences and clock drifts [14]). This is realistic for the set of targeted real-time automation applications, because of their implementation on dedicated systems.

• Communication, provided by the BSL at level 5 of the OSI protocol stack, is assumed to be perfect (no lost messages, no duplicates, keeping message order). In order to increase the coverage for this assumption, a set of mechanisms can either be deployed or developed at the lower OSI levels; DepAuDE relies on the Ethernet CRC error detection mechanism and level 2 retransmission mechanisms. For the pilot application from section 3, UDP/IP over a switched Ethernet network was adequate; for other situations TCP/IP might prove better if real-time constraints are fulfilled.

• As the communication mechanism targets groups of tasks, there is an OSI level 5 multicast service, whose behavior is assumed to be atomic. If this assumption coverage is too low, dedicated atomic multicast support and group membership functions can be added.

The DepAuDE middleware supports the reintegration of BSL and FTM, and can reload application tasks. The application in itself is responsible for reintegrating these restarted tasks into the ongoing execution, as no checkpoint/restore mechanisms are included.
3 Case study

The Local Control Level module (LCL) is a PSAS component providing control and protection functions for the primary substation, as well as an interface to the operator and -over the inter-site network- to remote control systems and remote operators [2]. The LCL controls the switches to the two HV/MV transformers, the switch connecting the Red MV bar (on the left) to the Green MV bar (on the right), as well as switches local to the MV lines (Figure 1). The pilot application implements two functions from the LCL module: automatic power resumption (function1) and parallel transformers (function2).

Function1 allows automatic power resumption when a HV/MV transformer goes down, e.g. triggered by internal protection (temperature too high, oil alarm, …). It disconnects the MV lines connected to the busbar of the transformer, computes the load carried by the transformer just before the event happened, and if possible, causes the remaining transformer to take the entire load, as e.g. in the following scenario:

- (Initially) Red transformer carries 32 MVA (8 lines of 4 MVA) and Green transformer 24 MVA (8 x 3 MVA);
- (Anomaly) An internal protection mechanism shuts down the Green transformer, and its power drops from 24 MVA to zero. The switch connecting the Green bar to the Green transformer opens. (The switch connecting the Red bar to the Red transformer remains closed and the switch connecting the two bars remains open.)
- (Reaction) The switch connecting the Green bar to the Red bar receives the command to close. It closes 1 execution cycle (100 ms) later and the load carried by the Red transformer rises to 56 MVA.

Function2 (parallel transformers) consists of a series of automatic actions, assisting remote operators. E.g., an operator can request to switch on a transformer and function2 translates this request into a specific sequence of commands. Such a re-insertion scenario may be applied some time after transformer exclusion.

3.1 System setup

The PSAS application has been developed using a proprietary, automata-based, design environment based on the specification technique ASFA [11]. Application development consists of several steps:

- Function1 and function2 are extracted from the PSAS application [2] and specified through the ASFA Graphical Editor [1], obtaining a tabular description of the pilot application.
- These ASFA tables are processed by the ASFA-C Translator [13], producing a target-independent C-code version of the application, and by the ASFA Partitioner, allowing an application to be mapped to a single task or decomposed into a set of tasks [4]. The single task version has been used for the functional test of the application on a single host node, while a four-task version was selected for testing on a distributed system.
- At run time, the Distributed Execution Support Module, composed of BasicSoftware (BSW) and Executive, enforces cyclic execution, typical for PLC-based automation systems (PLC=Programmable Logic Controller). Robust execution is ensured by cyclically refreshing the I/O image and the non-protected memory areas, while the application’s state is safeguarded by hardware or software mechanisms [5]. The BSW takes care of synchronization and exception handling, while the Executive supplies the RTOS interface and a set of ASFA-specific library functions.

A peculiarity of the ASFA environment is that the application code is automatically obtained by translating the automata-based specification [11]. Besides reducing the probability of introducing coding errors, this approach provides portability to all platforms supported by the Distributed Execution Support Module.
As shown in Figure 2, this pilot application was deployed on a distributed system consisting of three dedicated heterogeneous (“target”) processors for the automation functions and two standard PCs for support functions, interconnected by an Ethernet switch:

- N1 and N2: two industrial PCs (VMIC and INOVA), with VxWorks as RTOS;
- N3: Siemens SIMATIC M7: an extended PLC with I/O modules, with RMOS32 as RTOS;
- N4: Linux-based standard PCs, hosting the Backbone;
- N5: Windows-NT PC with Operator Console functions. For inter-site connections (not considered here), an additional node hosts the gateway software.

The pilot application runs on this heterogeneous hardware equipment; input and output from/to the field is simulated. Synchronization signals, for cyclic application execution, are generated by the internal clock of one of the nodes (in a real set-up, they are obtained from an independent, external device). The following assumptions are made for the target nodes:

- All three target nodes (N1, N2 and N3 in Table 1) are attached to I/O components on the field (PU = Peripheral Units on Figure 1).
- The target node N3 handles the synchronization signal. In order to provide a backup solution in case of fault on N3, synchronization interrupts are also available at N1 and N3.

### 3.2 Instantiating DepAuDE on PSAS

The run-time components of the DepAuDE framework are integrated into the PSAS pilot application (see Table 1). The fault containment region is a node.

- An RMOS32 and a VxWorks implementation of the BSL tasks run on the target nodes (N1, N2, N3); a Linux and WinNT version runs on the host nodes (N4, N5).
- A LAN Monitor – an FTM used for detecting crashed or isolated nodes – is present on all nodes.
- The BackBone task, executing recovery strategies, is allocated to N4.
- Each of the three target nodes hosts an instance of the ASFA Distributed Execution Support Module, composed of BSW and Executive. Each instance of the BSW is able to act as master (BSW_M, on the master node) or slave (BSW_S, on the slave nodes). The role is chosen depending on the specific system configuration. All BSW_S make up the BSW_SLAVE_GROUP. The configuration with highest performance (see below) requires BSW_M to be allocated to N3 and the BSW_S processes to run on N1 and N2. Executive process instances are identical on each processing node and they compose the EXECUTIVE_GROUP.

The allocation of the application tasks depends on the partitioning of the two LCL functions (function1 and function2), among which there is no communication. Function2 consists of a single task, PARALLEL_TRS, function1 (automatic power resumption) consists of three tasks: two tasks (BUSBAR1 and BUSBAR2) handle low-level, I/O dependent, computations relative to the MV lines attached to each busbar; one task, STRAT, coordinates the whole function and performs no field I/O. There is no communication between the two BUSBAR tasks, while both communicate with STRAT.

The basic constraint for allocating tasks to nodes is that a task that controls a specific plant component should be allocated to a processor attached to that plant component (due to I/O paths). As both functions of the pilot application control the same set of field components (same transformers and switches), all target nodes are assumed to be connected to that portion of the field. We assume that target node N2 provides better computing performance than N1.

The start-up configuration is the optimal distribution of application tasks onto the heterogeneous hardware. The most performant configuration, Config_0 in Table 2, does not require off-node communication among the application tasks:

---

**Table 1: Allocation of middleware tasks to nodes**

<table>
<thead>
<tr>
<th></th>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>N4</th>
<th>N5</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSL tasks</td>
<td>*</td>
<td>*</td>
<td>✓</td>
<td>*</td>
<td>✓</td>
</tr>
<tr>
<td>Backbone</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>LAN Monitor</td>
<td>*</td>
<td>*</td>
<td>✓</td>
<td>*</td>
<td>✓</td>
</tr>
<tr>
<td>Operator Console</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSW</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Executive</td>
<td>✓</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

---

**Figure 2: PSAS hardware architecture**
Table 2: Different configurations to allocate active PSAS application tasks to target nodes

<table>
<thead>
<tr>
<th>Config</th>
<th>N1</th>
<th>N2</th>
<th>N3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>PARALLEL_TRS</td>
<td>STRAT, BUSBAR1, BUSBAR2</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>CRASHED</td>
<td>PARALLEL_TRS</td>
<td>STRAT, BUSBAR1, BUSBAR2</td>
</tr>
<tr>
<td>2</td>
<td>PARALLEL_TRS</td>
<td>CRASHED</td>
<td>STRAT, BUSBAR1, BUSBAR2</td>
</tr>
<tr>
<td>3</td>
<td>STRAT, BUSBAR1, BUSBAR2</td>
<td>-</td>
<td>CRASHED</td>
</tr>
</tbody>
</table>

- no application task is allocated to N3, whose BSW acts as master and handles communication with the remote control center;
- PARALLEL_TRS runs on N1;
- BUSBAR1, BUSBAR2, and STRAT are allocated to N2.
- Each application task has at least one standby replica task_Ri on a different target node Ni (i=1..3).

### 3.3 PSAS recovery strategy

In order to cope with temporary and permanent physical faults affecting the information and communication infrastructure of the PSAS, an appropriate recovery strategy has been designed and coded as a set of Ariel recovery scripts. It combines different kinds of error detection mechanisms, error recovery and system reconfiguration. Reconfiguration is statically associated to the crash of a single node. If two nodes crash simultaneously no reconfiguration is possible. The following scripts are examples of recovery actions.

**Example 1.** If a slave node (e.g., N1) crashes, the LAN Monitor detects this event and notifies the Backbone executing the following Ariel code:

``` Ariel
IF  
    PHASE(TASK{BSW_M}) == {NEW_CYCLE_PH}] 
THEN  
    ISOLATE NODE[N1]  
    SEND {CONFIG_1} TASK{BSW_MSG_M}  
    SEND {CONFIG_1} GROUP{BSW_SLAVE_GROUP}  
    RESTART GROUP{EXECUTIVE_GROUP}  
    RESTART TASK{PARALLEL_TRS_R2}  
FI
```

If the guard of the above script is fulfilled, application tasks are reconfigured as CONFIG_1 from Table 2. CONFIG_1 maintains the full PSAS functionality by transferring PARALLEL_TRS to N2, actually activating its spare replica. This node is able to cope with the whole computational load, as it does not need to perform communication requested by the BSW_M’s functions. To avoid undesired interference by the Backbone during critical phases of BSW_M activity, a condition on the current execution phase (PHASE(TASK{BSW_M}) == {NEW_CYCLE_PH}) must be satisfied in conjunction with the crash test. The ISOLATE NODE action corresponds to informing other nodes that they may not accept any message from the isolated peer -even if it comes alive again- until the isolation is undone.

**Example 2.** If a target node (e.g. N2) crashes during a different execution phase of the master BSW, then this error is notified by the BSW_M to the Backbone (through RaiseEvent(RE_BSW_error)), causing the execution of the following ARIEL code:

``` Ariel
IF  
    [EVENT {RE_BSW_error}]  
THEN  
    IF  
        [FAULTY NODE[N2] AND RUNNING NODE[N3]] 
    THEN  
        ISOLATE NODE[N2]  
        SEND {CONFIG_2} TASK{BSW_MSG_M}  
        SEND {CONFIG_2} TASK{BSW_MSG_S1}  
        RESTART GROUP{EXECUTIVE_GROUP}  
        RESTART TASK{BUSBAR1_R3}, TASK{BUSBAR2_R3}, TASK{STRAT_R3}  
        RESTART TASK{PARALLEL_TRS_R1}  
    FI
FI
```

Hence the system is reconfigured as Config_2: the spare replicas of BUSBAR1, BUSBAR2 and STRAT are activated on N3.

**Example 3.** In case of a fault on target node N3 (where BSW_M is running), the following ARIEL code is executed, triggered by error detection by the LAN Monitor and subsequent notification to the Backbone:

``` Ariel
IF  
    [FAULTY NODE[N3] AND RUNNING NODE[N1] AND RUNNING NODE[N2]]  
THEN  
    ISOLATE NODE[N3]  
    SEND {CONFIG_3} GROUP{BSW_SLAVE_GROUP}  
    SEND {BACKUP_MASTER} TASK{BSW_MSG_S2}  
    RESTART GROUP{EXECUTIVE_GROUP}  
    STOP TASK{PARALLEL_TRS}  
    RESTART TASK{STRAT_R1}, TASK{BUSBAR1_R1}, TASK{BUSBAR2_R1}  
FI
```

Hence, the function of master node is transferred to N2 and the application tasks of N2 are moved to N1. As N1 cannot support both application functions simultaneously, PARALLEL_TRS is disabled, thus proceeding to a graceful degradation of the automation system (config_3).

**Evaluation.** Other recovery strategies, such as resuming all tasks on a node after a transient fault, or shutting down the system when reconfiguration is not possible, have also been coded in ARIEL and implemented. We did not provide recovery strategies associated with a crash of N4 or N5, because they are not target nodes, they are not concerned with the automation control function itself; so even if they crash, the application is not endangered. In a real deployment they could be replicated or could backup each other.

Figure 3 shows the user interface of the pilot application demonstrator.
4 Summary and lessons learned

The lack of flexibility that is inherent to dedicated hardware-based fault tolerance solutions makes their adoption not cost-effective in cases where similar functionality has to be deployed in several sites, each characterized by a slightly different environment. This paper presented the integration of the DepAuDE architecture into the distributed automation system of a primary substation. The deployment of this fault tolerance middleware allows different recovery strategies to be integrated on a heterogeneous platform. Given the generality of the methods and techniques used the designed solution is applicable to a wide class of process automation systems. Following points summarize the lessons learned:

- The ASFA design environment with automatic code generation provides several advantages: less development time, absence of coding errors, portable application code and possibilities for application partitioning. It is straightforward to interface it to IEC 61850-compliant Intelligent Electronic Devices (IED).
- The implementation effort required to integrate the DepAuDE BSL into an ASFA application was limited (about 2400 lines of code for the RMOS and VxWorks targets). The communication mechanism supplied by the DepAuDE BSL provided transparent inter-process communication among ASFA application tasks. The grouping of tasks revealed useful when implementing the standby replicas. Inter-processor communication among application tasks strongly influences application performance and reconfiguration time in case of faults. Therefore inter-processor data flow should be avoided if possible, or at least minimised.
- The deployment of the DepAuDE middleware allowed integrating several recovery strategies on a heterogeneous platform. The separation between functional and error recovery programs provides flexibility to modify recovery strategies without requiring major modifications to the application, while tolerating the same physical faults as in the dedicated hardware solutions.

Acknowledgements. This project has been partially supported by European project IST-2000-25434 (www.DepAuDE.org), the K.U.Leuven Research Council (GOA/2001/04), and the Fund for Scientific Research - Flanders (Belgium, F.W.O.) through the Postdoctoral Fellowship for Geert Deconinck and “Krediet aan Navorers” 1.5.148.02.

5 References