FAULT TOLERANCE IN INTEGRATION INTERFACES OF BUSINESS SOFTWARE

Rivo LEMMIK 1, Dr. Kristo KARJUST 2, Dr. Tauno OTTO 3

Department of Machinery, Tallinn University of Technology, Tallinn, ESTONIA
1 PhD Student, rivo.lemmik@ttu.ee, 2 Prof assoc Dr., 3 Prof Dr.

ABSTRACT

Current paper describes different integration technologies of business software to analyze fault tolerance and reliability of those interfaces. Main focus is on the ERP (Enterprise Resource Planning) software Microsoft Dynamics NAV, which can be integrated with other business applications as central business data source. To highlight real-life situations the sample integration project is used as a case-study, where different fault tolerance levels are used for integration interfaces.

Keywords: Integration Interfaces, Fault Tolerance, Business Software, Enterprise Service Bus, Web Services

1. INTRODUCTION

The rapidly increasing competitiveness in market highlights the importance of design quality, maximizing productivity, multi-company collaboration, optimal price levels and predictability. The main focus of the manufacturer is to innovate, get products to the market faster, reduce errors and increase flexibility. The manufacturers have been continuing to improve their products, information systems developments and management abilities. Because of that in the past years have seen growing investments in the area of product lifecycle management (PLM) [1, 2], enterprise resource planning (ERP), real time monitoring and optimization [3,4], and integration technologies of business software and information systems [5].

On the other side there are increasing demands to make the application software systems more tolerant to faults. From a user's point of view, fault tolerance has two dimensions: availability and data consistency of the application. For example users of telephone switching systems demand continuous availability, whereas bank teller machine customers demand the highest degree of data consistency [6].

Exception handling is one application level feature to interrupt normal operation to handle abnormal responses. In the context of software fault tolerance, exceptions are signaled by the implemented error detection mechanisms as a request for initiation of an appropriate recovery. The design of exception handlers requires that consideration is given to the possible events triggering exceptions, the effects of those events on the system, and the selection of appropriate mitigating actions [7].

Application level fault tolerance for integration interfaces contains different layers to monitor services and handle exceptions. Automated integration interfaces for ERP software MS Dynamics NAV are mainly based on the NAS (Navision Application Server) windows service which operates a number of server-side background processes. To ensure reliability of those processes the following is required:

- Monitor that NAS service is running by using windows standard monitoring software
- Monitor that NAS service is alive by communicating one specially designed interface
- Global framework for processes to handle unexpected errors to keep processes alive
- Per process error handling in database transaction level for continuous processing
- Per transaction error handling to process through all known exceptions
- Watchdog process to send out notifications about exceptions and error situations
- Provide application user interface to access processing and error logs
- Provide mechanism to make required corrections and initiate recurrent processing

2. APPLICATION LEVEL FAULT TOLERANCE

Availability and data consistency in an application are traditionally provided through fault tolerant hardware and operating system, used by the application for its execution. New trends are emerging in the marketplace that is changing this tradition. Standard commercial hardware and operating systems are becoming highly
reliable, distributed and inexpensive to the extent that they are now off-the-shelf commodity items. New application software systems are increasingly networked and distributed, i.e. mostly client-server systems [6].

Many of those applications are also built from reusable components whose sources are unknown to the application developers. Due to this complexity in application software, the proportion of failures due to faults in the application software is increasing. The End-to-End type of arguments implies that one needs fault tolerance in the application software itself to handle such failures [6].

2.1. Application Infrastructure

Hardware level is the lowest level to catch application errors. If an error passed through all the higher levels then it will cause assembler error on microprocessor level and whole system need to be restarted.

Operating System level can catch application errors which haven’t been handled at application software or database engine levels. An error at this level can cause restart of the application software or database engine service.

Database Engine level can ensure data consistency per transaction. Database transaction will be initiated and finished by application software and it can contain several data manipulation actions. If application software dies or loses database connection during the transaction then database engine can automatically initiate rollback for whole transaction after transaction timeout was occurred to avoid inconsistent data state.

Application Software level error handling can detect erroneous processes and kill them to continue with other processes if possible. In some situations this level can cause that application continues working in abnormal state and finally can collapse and need to be restarted.

2.2. Application Exception Handling

Exception handling level is the best level for reliable error handling to avoid errors handled at infrastructure level. This level is highly application business logic specific and can react more adequately for errors than infrastructure level. On the other hand it is also more costly to build very deep error handling at this level because of each upper layer at this level can be less unified and will be more business logic specific.

Try/Catch methods should be used to enclose some sort of business logic to catch irregular or unexpected errors to handle those and continue processing as defined. This layer can try to repeat some function several times if an error was irregular or if it was associated with some external problem like network errors etc. Also it is possible to build regular error handling at this layer for known but rare situations where erroneous situation can be skipped or function can be cancelled with error message.

If/Then methods are highly application business logic specific and should be used in situations if there are different alternative workflows possible for exceptional cases. Also this layer is commonly used to validate data input according to the data validation rules.

User Interface layer is the highest application layer which can be used for basic data validation to avoid processing of invalid data input and to output instant warnings or error messages for user.

3. APPLICATION INTEGRATION INTERFACES

SOA is concerned with the independent construction of services which can be combined into meaningful, higher level business processes within the context of the enterprise.
Typical “Spaghetti” Enterprise Application Integration where adapters interface with applications at their integration points is presented in Figure 2.

A Service Oriented Architecture presented in Figure 3 describes several aspects of services within an enterprise application: [8]

- The granularity and types of services
- How services are constructed
- How the services communicate at a technical level
- How the services are combined together (i.e. orchestrated)
- How the services interoperate at a semantic level (i.e. how they share common meanings)
- How services contribute to IT and Business Strategy

![Figure 3. SOA Enterprise Application Integration [8]](image)

Based on SOA the special Service Oriented Integration (SOI) methodology is evolved:

- An architectural and technology based approach to exposing and integrating existing applications as services
- Builds on EAI technology, using new Web services based platforms
- Exposes services to a bus, not point-to-point
- Extends SOA to integration solutions

Benefits of SOI Approach presented in Figure 4. [8]

- Each system is integrated once into the service bus, rather than many times for each point-to-point connection
- Multiple services can be easily constructed from the integration of existing applications
- New processes can be constructed from the service
- Layered SOI approach enables quickly reconfiguring processes or services without needing to change operational systems
- Layered SOI approach allows operational systems to change without affecting business processes

The Enterprise Service Bus (ESB) provides an ideal platform for SOI applications and SOI combines Web service, EAI and SOA.

### 3.1. Three-Tier Architecture

The three-tier architecture model, which is the fundamental framework for the logical design model, segments an application's components into three tiers of services. These tiers do not necessarily correspond to physical locations on various computers on a network, but rather to logical layers of the application. How the pieces of an application are distributed in a physical topology can change, depending on the system requirements [9].

**The presentation tier**, or user services layer, gives a user access to the application. This layer presents data to the user and optionally permits data manipulation and data entry. The two main types of user interface for this layer are the traditional application and the Web-based application. Web-based applications now often contain most of the data manipulation features that traditional applications use. This is accomplished through use of Dynamic HTML and client-side data sources and data cursors [9].

**The middle tier**, or business services layer, consists of business and data rules. Also referred to as the business logic tier, the middle tier is where developers can solve mission-critical business problems and achieve major productivity advantages. The components that make up this layer can exist on a server machine, to assist in resource sharing. These components can be used to enforce business rules, such as business algorithms and legal or governmental regulations, and data rules, which are designed to keep the data structures consistent within either specific or multiple databases. Because these middle-tier components are not tied to a specific client, they can be used by all applications and can be moved to different locations, as response time and other rules require. For example, simple edits can be placed on the
client side to minimize network round-trips, or data rules can be placed in stored procedures [9].

The data tier, or data services layer, interacts with persistent data usually stored in a database or in permanent storage. This is the actual DBMS access layer. It can be accessed through the business services layer and on occasion by the user services layer. This layer consists of data access components (rather than raw DBMS connections) to aid in resource sharing and to allow clients to be configured without installing the DBMS libraries and ODBC drivers on each client [9].

The data tier, or data services layer, interacts with persistent data usually stored in a database or in permanent storage. This is the actual DBMS access layer. It can be accessed through the business services layer and on occasion by the user services layer. This layer consists of data access components (rather than raw DBMS connections) to aid in resource sharing and to allow clients to be configured without installing the DBMS libraries and ODBC drivers on each client [9].

During an application's life cycle, the three-tier approach provides benefits such as reusability, flexibility, manageability, maintainability, and scalability. You can share and reuse the components and services you create, and you can distribute them across a network of computers as needed. You can divide large and complex projects into simpler projects and assign them to different programmers or programming teams. You can also deploy components and services on a server to help keep up with changes, and you can redeploy them as growth of the application’s user base, data, and transaction volume increases. [9]

3.1.1. Database tier integration

Database level integration interfaces were common in Two-Tier applications where business logic was mostly located at database level. This integration level is based on database tables, views and stored procedures and integrated applications will connect directly into each other databases to query and manipulate the data.

For applications with Three-Tier architecture this level isn’t often used for integration because of the database level doesn’t contain any more business logic and it’s easy to break data consistency by manipulating data directly at database level. As well the technology of the integration interface at this level is database engine specific and there can be several complexity or compatibility issues between different platforms.

3.1.2. Middle tier integration

For applications with Three-Tier architecture this level is commonly used to build integration interfaces. Actually there isn’t big difference at all to provide Middle Tier services for its own presentation layer or for some other application. As integration technology the SOAP Web Services are commonly used at this level which are mostly cross-platform compatible and widely supported.

SOAP Version 1.2 (SOAP) is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. It uses XML technologies to define an extensible messaging framework providing a message construct that can be exchanged over a variety of underlying protocols. The framework has been designed to be independent of any particular programming model and other implementation specific semantics. The ability to use SOAP in a particular environment will vary depending on the actual constraints, choice of tools, processing model, or nature of the messages being exchanged. SOAP has been designed to have a relatively small number of dependencies on other XML specifications, none of which are perceived as having prohibitive processing requirements. Also, limiting use of SOAP to small messages instead of arbitrarily-sized messages and supporting only a few specific message types instead of implementing generalized processing could significantly lower processing requirements [10].

During an application's life cycle, the three-tier approach provides benefits such as reusability, flexibility, manageability, maintainability, and scalability. You can share and reuse the components and services you create, and you can distribute them across a network of computers as needed. You can divide large and complex projects into simpler projects and assign them to different programmers or programming teams. You can also deploy components and services on a server to help keep up with changes, and you can redeploy them as growth of the application's user base, data, and transaction volume increases. [9]

3.1.1. Database tier integration

Database level integration interfaces were common in Two-Tier applications where business logic was mostly located at database level. This integration level is based on database tables, views and stored procedures and integrated applications will connect directly into each other databases to query and manipulate the data.

For applications with Three-Tier architecture this level isn’t often used for integration because of the
As communications protocols and message formats are standardized in the web community, it becomes increasingly possible and important to be able to describe the communications in some structured way. WSDL addresses this need by defining an XML grammar for describing network services as collections of communication endpoints capable of exchanging messages. WSDL service definitions provide documentation for distributed systems and serve as a recipe for automating the details involved in applications communication. A WSDL document defines services as collections of network endpoints, or ports. In WSDL, the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. This allows the reuse of abstract definitions: messages, which are abstract descriptions of the data being exchanged, and port types which are abstract collections of operations. The concrete protocol and data format specifications for a particular port type constitute a reusable binding. A port is defined by associating a network address with a reusable binding, and a collection of ports define a service [11].

3.1.3. Presentation layer integration

Integration interfaces at presentation layer were more often used in Two-Tier applications where data validation rules were mostly located at this level. Integration interfaces at presentation layer can treat more like data export tools and data mass manipulation tools above of the application user interface. As integration technology CSV or Excel file import/export is commonly used at this integration level.

For applications with Three-Tier architecture this level is used in modern user interfaces to support simple copy/paste data transfer between application software and office software e.g. Microsoft Excel and Microsoft Word. Commonly data export from application software towards office software is supported and also sometimes data transfer back from office software is supported as well.

3.2. Microsoft Dynamics NAV

Microsoft Dynamics NAV is an enterprise resource planning (ERP) software product from Microsoft. The product is part of the Microsoft Dynamics family, and intended to assist with finance, manufacturing, customer relationship management, supply chains, analytics and electronic commerce for small and medium-sized enterprises. Value-added resellers (VARs) can have full access to the business logic source code, and it has a reputation as being easy to customize. For modifications of the system, the proprietary programming language C/AL is used. [12]

Figure 7. MS Dynamics NAV 2009 R2 RoleTailored Client [12]

The product itself has gone through several name changes as the original Navision Company or Microsoft has tried to decide on how it should be marketed. The names "Navision Financials", "Navision Attain", "Microsoft Business Solutions Navision Edition", and the current "Microsoft Dynamics NAV" have all been used to refer to this product. In December 2008 Microsoft released Dynamics NAV 2009, which contains both the original "classic" client, as well as a new three-tier GUI called the RoleTailored Client (RTC). Microsoft originally planned to develop an entirely new ERP system (Project green), but has decided to continue development of all ERP systems (Dynamics AX, Dynamics NAV, Dynamics GP and Dynamics SL). All four ERP systems will be launched with the same new role based user interface, SQL based reporting and analysis, SharePoint based portal, Pocket PC based mobile clients and integration with Microsoft Office. [12]

Relative to Microsoft’s other 3 ERP products, Dynamics NAV’s sector is distribution and manufacturing companies that want more than “out of the box” functionality. The solution has a standard feature set, but it can also be thought of as an “ERP System construction set”. The Pascal-like development language is easily accessible to appropriate developers and is designed for rapidly customizing the software. There is no need for complex server side Transact-SQL stored procedures as the one language manages the application and database. [12]

According to Microsoft, Dynamics NAV is being used by approximately 65,000 companies, with over 1.3 million end user licenses. Approximately 13,000 of those companies are in the US. It was the first and only mid-market ERP application to break one-million end user licenses. Microsoft Dynamics NAV delivers integrated functionality to provide support for: [12]

- Financial management
- Supply chain management
- Manufacturing
• Distribution
• Customer relationship management
• Sales and marketing
• Service management

With Microsoft Dynamics NAV 2009, the architecture has changed from a two- to three-tier architecture, enabling the new RoleTailored client and Web services. Web services are a standard widely used method for integrating applications and are supported in Microsoft Dynamics NAV 2009. By implementing Web services, you can access Microsoft Dynamics NAV data and business logic from outside the product in a standard, secure way. This enables you to connect Microsoft Dynamics NAV to other systems within an organization.

4. EAS CASE STUDY

As a case study in current paper one NAV project is described where NAV is implemented as a central IT solution of the organization and it has lot of internal and external integration interfaces with other applications and IT systems. It’s highly customized solution developed for Enterprise Estonia (EAS) to cover all the main business processes in the organization.

Due to the solution has been developed and grown during 8 years there are used different architectural and technological solutions. Main external integrations are:

• Ministry of Finance – SOAP Web Services
• State treasury – X-Road Web Services
• Business Register – Direct XML Interface
• Certification Center – SOAP and COM Library

There are three main parts that IT system consists and which are deeply integrated with each other in-house by using following technologies:

• MS Dynamics NAV as central processing solution
  o SOAP Web Services
  o X-Road Web Services
  o Direct XML Interface
  o Windows COM Library
  o Microsoft Message Queuing
  o Database Views and Stored Procedures
  o CSV file import/export

• MS SharePoint as document management system
  o SOAP Web Services
  o Database Views and Stored Procedures

• .NET Web Portal as Customer Self-Service Centre
  o SOAP Web Services
  o Microsoft Message Queuing

All of those three main parts can be divided into modules which have internal integrations with each other. Following main functional areas are covered:
• Customer Relationship Management (CRM)
• Proceeding of Projects (PAM)
• Procurement Management
• Processing of Purchase Invoices
• Finance Management

As it is very complex and extensive IT solution which covers very critical business areas it needs reliable error and exception handling solutions for the integration interfaces.

Error handling and monitoring of those interfaces is divided into several levels to acquire optimal combination of reliability and cost efficiency.

4.1. Service Level Monitoring

This level is used to keep middle tier services running and alive in any unexpected error situations. Normally working system shouldn’t cause any errors at this level and all the errors occurred this level need to be analyzed case by case to avoid them at the future.

• Monitor that NAV services are running by using windows standard monitoring software. Service states and error messages are logged to Windows Event Log.
• Monitor that NAV services are alive by communicating them periodically through specially designed interface. Service states and error messages are logged to Windows Event Log.
• Monitor Windows Event Log messages by using windows standard monitoring software. According to specified message type and importance initiate specified notifications or actions.
• If service has stopped then try to restart it. If automatic restart fails then send out system critical state notification to administrators.
• Send out non-critical notifications about occurred error and warning situations to maintenance team.
• If an error at this level occurred during integration process e.g. during SOAP request then process terminates and integration interface can’t send out any response. This means that other system needs to wait until timeout occurs.

4.2. Process Level Error Handling

This level is used to catch known but rare errors which don’t need to be handled according to the business logic. This level has unified design by using try/catch methods to enclose all processes. If error occurs then service is skipping erroneous process and will continue with next processes. In the situation where erroneous process was preceded for some next process the next process probably will get the transaction level error. Errors occurring at this level commonly are associated with invalid web service request or invalid input data. Responses sent out at this level are mainly general error messages e.g. SOAP protocol errors.

• Main part at this level is global and unified framework for processes to handle errors to keep services running and alive. Commonly this framework uses try/catch methods to enclose processes.
• Error handling is process-oriented with the purpose to not affect other processes. If multiple processes are connected at business logic level then those relations aren’t identified.
• Main purpose at this level is to skip erroneous processes and continue with next processes and to avoid infinite loops of the erroneous process.
• Global error logging into application level error logs to analyze errors and if necessary then manually initiate recurrent processing.
• Send out notifications according to message types and importance settings.
• If an error at this level occurred during integration process e.g. during SOAP request then unhandled exception or SOAP protocol error response will be sent out.

4.3. Transaction Level Exception Handling

Exception handling at this level is most advanced and costly but also most important to keep system running smoothly and reliably. This level contains exception handling rules at the business logic layer by using if/then methods. All known possible exception situations are handled by using alternative workflows. It means that exceptions aren’t just skipped but some alternative way on workflow can be used to continue positive processing of the transaction. All known and possible error situations can also initiate some alternative workflow or also cancel the transaction with predefined error codes. Errors and exceptions at this level commonly are associated with data quality and data validation rules according to associated business logic. Responses sent out at this level are predefined and detail error messages which can be associated as well with the other systems business logic and workflows to define how to proceed most adequately in the situation where an error was occurred at this level.

• Error handling is transaction-oriented to process through all known exception situations without affecting other transactions in the same process.
• This level is based on business rules and alternative workflows defined for all possible known error and exception cases.
• Transaction specific issue logging into application level error logs to analyze cases and if necessary then
make data corrections and initiate recurrent processing.
• Send out notifications according to message types and importance settings.
• If an error at this level occurred during integration process e.g. during SOAP request then response with predefined error/status code will be sent out to integration interface.

4.4. User Level Intervention

This level is used for corrective and preventive actions to maintain the system and initiate system improvement tasks to avoid recurring cases hereafter.
• Provide user level mechanisms to react for notifications.
• Provide application level user interface to access all processing and error logs.
• Provide mechanism to make necessary data corrections and initiate recurrent processing.
• If process level error handling fails for some reason then terminate abnormal or looping processes.
• Continuously analyze occurred issues and initiate system improvement tasks.

5. CONCLUSION

In conclusion the error handling for integration interfaces is very important to build reliable and fault tolerant solutions. Compared to application user interfaces, where human can react for error situations creatively the integration interfaces without error handling will just stop working or continue working at inconsistent or looping state which can cause lot of problems. In worst cases also data damages or massive data avalanche or spamming can occur.

Very critical is to divide error handling into described levels adequately and reasonably. It is important to avoid service level errors in any case. To make decisions between process and transaction levels. It’s important to calculate optimal cost efficiency because of the transaction level is most costly, however process level is sometimes too broad.

6. ACKNOWLEDGEMENTS

This research was supported by:
• European Social Fund’s Doctoral Studies and Internationalisation Programme DoRa, which is carried out by Foundation Archimedes.
• ETF grants 8485 and 7852, targeted financing project SF0140035s12
• Innovative Manufacturing Engineering Systems Competence Centre IMECC (supported by Enterprise Estonia and co-financed by the European Union Regional Development Fund, project EU30006).

7. REFERENCES