COMPUTING
SCIENCE

# Architecting Holistic Fault Tolerance

Rem Gensh, Ashur Rafiev, Alexander Romanovsky
CSR
Newcastle University
Newcastle upon Tyne, UK

Alessandro Garcia
Informatics Department
PUC-Rio
Rio de Janeiro, Brazil

Fei Xia, Alex Yakovlev
School of EEE
Newcastle University
Newcastle upon Tyne, UK

**TECHNICAL REPORT SERIES**

**No. CS-TR-1505 November 2016**

**TECHNICAL REPORT SERIES**

---

**No. CS-TR-1505 November, 2016**

Rem Gensh, Ashur Rafiev, Alexander Romanovsky,
Alessandro Garcia, Fei Xia, Alex Yakovlev.

## Abstract

The optimality and maintainability of fault tolerance mechanisms in a computer system has typically not been a major topic of concern, mostly because fault tolerance is a non-functional system requirement. This paper proposes a Holistic Fault Tolerance architecture, based on a centralised fault tolerance management, with related functionality distributed across the entire system. The most suitable error detection and error recovery strategies for a given application are chosen by a special crosscutting controller depending on error rates, system performance and resource utilisation requirements. We discuss the motivation for introducing this holistic fault tolerance architecture and reason about its benefits from the point of view of optimal system operation and improved maintainability. The advantages and possible implementation challenges of the proposed approach are demonstrated by a real-world application.

**Bibliographical Details**

Rem Gensh, Ashur Rafiev, Alexander Romanovsky
CSR
Newcastle University
Newcastle upon Tyne, UK

Alessandro Garcia
Informatics Department
PUC-Rio
Rio de Janeiro, Brazil

Fei Xia, Alex Yakovlev
School of EEE
Newcastle University
Newcastle upon Tyne, UK

NEWCASTLE UNIVERSITY
Computing Science. Technical Report Series. CS-TR-1505

## Abstract

The optimality and maintainability of fault tolerance mechanisms in a computer system has typically not been a major topic of concern, mostly because fault tolerance is a non-functional system requirement. This paper proposes a Holistic Fault Tolerance architecture, based on a centralised fault tolerance management, with related functionality distributed across the entire system. The most suitable error detection and error recovery strategies for a given application are chosen by a special crosscutting controller depending on error rates, system performance and resource utilisation requirements. We discuss the motivation for introducing this holistic fault tolerance architecture and reason about its benefits from the point of view of optimal system operation and improved maintainability. The advantages and possible implementation challenges of the proposed approach are demonstrated by a real-world application.

## About the authors

Rem Gensh is a PhD student and a Research Technician at SRS group in the School of Computing Science of Newcastle University, Newcastle-upon-Tyne, UK. He graduated Kyrgyz-Russian Slavic University, Kyrgyzstan, with honors in 2008. After a 6-years extensive industrial experience as a software developer and a team leader, he reallocated to research area and started his PhD in 2014. He is involved in EPSRC/UK PRiME project. His research interests include fault tolerance, energy-efficient software design and many-core architectures.

Ashur Rafiev is an RA on the EPSRC PRiME Program Grant. In this project he leads the development of the ArchOn modelling environment.

Alexander (Sascha) Romanovsky is a Professor in the Centre for Software and Reliability, Newcastle University. His main research interests are system dependability, fault tolerance, software architectures, exception handling, error recovery, system structuring and verification of fault tolerance. He received a PhD degree in Computer Science from St. Petersburg State Technical University and has worked as a visiting researcher at ABB Ltd Computer Architecture Lab Research Center, Switzerland and at Istituto di Elaborazione della Informazione, CNR, Pisa, Italy. In 1993 he became a postdoctoral fellow in Newcastle University, and worked on the ESPRIT projects on Predictable Dependable Computing Systems (PDCS), Design for Validation (DeVa) and on UK-funded projects on the Diversity, both in Safety Critical Software using Off-the-Shelf components. He was a member of the executive board of EU Dependable Systems of Systems (DSoS) Project, and between 2004 and 2012 headed projects on the development of a Rigorous Open Development Environment for Complex Systems

(RODIN), and latterly was coordinator of the major FP7 Integrated Project on Industrial Deployment of System Engineering Methods Providing High Dependability and Productivity (DEPLOY). He now leads work on fault tolerance in Systems of Systems within the COMPASS project and is Principal Investigator of Newcastle's Platform Grant on Trustworthy Ambient Systems.

Fei Xia is a Senior RA on the EPSRC PRiME Program Grant. Fei is with the EE School. His research interests include Asynchronous Data Communication. Asynchrnous System Design. Systems and Networks on Chip. Energy and Power in Computing.

Alex Yakovlev received D.Sc. from Newcastle University in 2006, and M.Sc. and Ph.D. from St. Petersburg Electrical Engineering Institute in 1979 and 1982. Since 1991 he has been at the Newcastle University, where he worked as a lecturer, reader and professor at the Computing Science department until 2002, and is now heading the Microelectronic Systems Design research group (http://async.org.uk) at the School of Electrical, Electronic and Computer Engineering. His current interests and publications are in the field of modeling and design of asynchronous, concurrent, real-time and dependable systems on a chip. He has published four monographs and more than 200 papers in academic journals and conferences, has managed over 25 research contracts.

## Suggested keywords

Architecture; Fault tolerance; Crosscutting concerns; Performance; Operation modes;

# Architecting Holistic Fault Tolerance

Rem Gensh, Ashur Rafiev, Alexander Romanovsky
CSR
Newcastle University
Newcastle upon Tyne, UK

Alessandro Garcia
Informatics Department
PUC-Rio
Rio de Janeiro, Brazil

Fei Xia, Alex Yakovlev
School of EEE
Newcastle University
Newcastle upon Tyne, UK

***Abstract*—The optimality and maintainability of fault tolerance mechanisms in a computer system has typically not been a major topic of concern, mostly because fault tolerance is a non-functional system requirement. This paper proposes a Holistic Fault Tolerance architecture, based on a centralised fault tolerance management, with related functionality distributed across the entire system. The most suitable error detection and error recovery strategies for a given application are chosen by a special crosscutting controller depending on error rates, system performance and resource utilisation requirements. We discuss the motivation for introducing this holistic fault tolerance architecture and reason about its benefits from the point of view of optimal system operation and improved maintainability. The advantages and possible implementation challenges of the proposed approach are demonstrated by a real-world application.**

***Keywords—architecture; fault tolerance; croscutting concerns; performance; operation modes;***

## I. Introduction

Faults and failures are unavoidable in computer systems. To prevent catastrophic consequences of these failures, computer systems must be both reliable and safe, ensuring overall system dependability. In addition, they should be optimal in terms of resource utilisation because performance and energy efficiency are important factors for the systems regardless of scale from embedded devices to data centres. It is also crucial to provide an easy way to maintain and modify the system components to decrease outage time, improve developer's understanding of the system and reduce associated costs. The same demands are applied to fault tolerance (FT) mechanisms of the system.

During system design a lot of effort is made to provide high cohesion and loose coupling of the system components [1]. This approach can be applied very smoothly for functional properties and for business logic, because the same functionality may be placed in one unit and intricate implementation details may be hidden from other units. However, when FT mechanisms of such a unit are hidden, it makes system-wide FT less understandable and optimisable. Pragmatically, such an approach causes components to be designed maximally safe with the costs not always contributing to system-wide FT, precluding the possibility of centralised monitoring and dynamic tuning of the system based on interplay between required performance, resource utilisation and reliability.

To deal with these issues we offer to apply the holistic fault tolerance (HFT) architecture, which allows developers to control the system FT in a global crosscutting manner. The vision of "Holistic" Fault Tolerance was proposed in our recent paper [2] where we presented a novel approach to system FT taking into consideration non-functional characteristics of the system, such as reliability, performance and energy efficiency. This approach assumes that the FT mechanisms across the entire system are managed by a central component, allowing the developer to reason about certain error detection and error recovery strategies at the system scale, and not at the scale of separate components. The HFT architecture does not imply the alteration of firmly established FT techniques [3]. In contrast, the HFT architecture demonstrates how these techniques can be applied for the design and implementation of more optimal computer systems by reasoning about the system FT holistically, rather than concentrating on individual components only. The reason to introduce the holistic approach is the facilitation of system modularity by the separation of crosscutting concerns, such as FT mechanisms of the system.

While the HFT approach is considered as general and not restricted by certain application domains, our main focus is given to the component-based software architectures, since they are more suitable for the scope of this article. Therefore, generally this type of software architecture will be assumed further in the paper.

The main contribution of this paper is a specific HFT architecture, which allows the designer to have centralised access to the FT functionality of the system and to tune non-functional properties like reliability, performance and resource utilisation. In addition, we consider a general method that can be applied to facilitate the design of the HFT architecture. The goal of this study is to demonstrate that HFT is able to monitor system states and dynamically adjust the entire system to achieve optimal resource utilisation without uncontrolled reliability deterioration.

In this paper we present further development of the HFT architecture that has been made since our previous work, consider all its elements in details and describe the techniques that could be applied for the implementation of the HFT architecture. The rest of the paper is organised as follows. Section 2 provides the motivation and introduces the research

Rem Gensh is supported by the School of Computing Science of Newcastle University and by PRiME project EP/K034448/1.

question. Background support for the chosen study is discussed in Section 3. Section 4 describes the HFT architecture and all of its parts. A practical way to apply the HFT approach and its benefits are demonstrated by a real-world example in Section 5. Concluding remarks are given in Section 6.

## II. Motivation

Our approach is called *holistic*, because we propose to consider an entire application during the design of the system's FT. We have chosen FT property as main part of the concept, since it is an important crosscutting concern of the system, which can affect other non-functional properties of the system.

We offer to use a centralised unit to manage FT across the system, because it is the most suitable place for the given functionality. Here we can analyse reliability, performance, and resource utilisation of the application and make necessary adjustments of the application based on the trade-off between these properties.

According to academic [4] and industrial [5] studies, FT is a crosscutting concern for computer systems. In this way, during the design and implementation of FT functionality the main focus should be made system-wide, rather than on components. It is more convenient to centralise FT-related code in order to improve the modularity of the system, since the relevant FT functionality will be coordinated by one module, unit or component, simplifying the understanding and access to the FT mechanisms. The main dilemma is how to create a crosscutting component-controller implementing system-wide FT scenarios and at the same time to avoid over-complicating the system architecture by binding this centralised controller to all crucial components of the system.

In previous work [2] we presented two reasons for introducing HFT. The first is optimal system-wide operation. Without a holistic approach a system may consist of a set of optimal components without global optimality. The other relates to system maintenance – system-wide FT is not easily understandable and modifiable without a holistic approach. These remain the fundamental motivations for this paper.

## III. Backgound and existing work

This section provides an analysis of the existing approaches to system structuring and FT management, and considers the following issues: a centralised management of FT, system architectures based on goal-seeking behaviour, modular FT architectures and operation modes.

### A. *Centralisation of FT management*

The study [6] provides the notion of guardian – a special global exception handler for a distributed system. In addition, they consider the implementation of distributed exception handling and global exception handling and analyse the provided guardian model. The goal of the guardian is to enhance existing exception handling models and provide the basis for them. Authors note that in a distributed systems exception handling is far different from sequential exception handling, since distributed systems require communication and coordination of exception handlers. Moreover, several exceptions may be raised concurrently. Thus, each participating process of the application should invoke the correct handler. In the guardian model, the correct handler for each process is chosen by the guardian according to the application defined recovery rules. This allows the guardian to orchestrate the recovery action of each involved process. Guardian model distinguishes global exceptions coordinated through the guardian and local exceptions, which are handled by those processes where they occurred. Main advantages of the guardian model are separation of global exception handling from participant processes and flexible primitive scheme for distributed exception handling. However, this approach involves scalability and performance overheads for the implementation of reliable broadcast with participating processes. The second limitation is a complexity of definition of the contexts and corresponding set of exceptions and handlers according to the guardian rules.

### B. *Systhems with goal-seeking behaviour*

Another interesting approach is to consider the system from the goal-achieving point of view. One example is described by Brooks as the architecture of layered control system developed for mobile robots [7]. A control system is able to execute many complex processing tasks in real time. Instead of decomposing the problem into functional units, the author decided to apply task achieving behaviour decomposition. Several mobile robot requirements were identified. Firstly, the robot has multiple goals sorted by priority because the goals could be conflicting. Secondly, for navigation purposes, the robot uses multiple sensors, which not always give very precise data. The third point is robustness. If some parts of the control system fails, the robot should rely on working components. Brooks defines his initial motivation stating that it is not necessary to use very complex control systems in order to achieve complex behaviour. Levels of competence and layers of control are applied to solve each small decomposed subproblem. Levels of competence are defined as a guide for this work. Lower levels implement simple behaviours like avoiding the objects and wander aimlessly without hitting the walls, etc. Each next level offers more complex behaviour and includes each earlier level of competence as a subset. For each level of competence there is a corresponding layer of control. Layers of control are added incrementally without changing the lower layers. A higher layer augments lower layers of the control system, but the lower layers still produce the results without knowing about the higher layers. The author calls this the subsumption architecture. Such an architecture provides additional robustness since the lower levels of competence are well debugged and continue to produce the results. If the higher level is unable to produce the result during the specified time, then the lower level will produce the acceptable result. In addition, new layers can be added later if the control system requires additional functionality. The given architecture does not require any central control, because the system is considered as a system of independent agents. However, the lower layers produce the results despite the fact that these results will not be used hereafter. Such a scheme lacks system-wide coordination. In some cases, this approach leads to overdesign: redundant operations or waste of the resources.

Another relevant work is the idea of the Teleo-Reactive programs presented by Nilsson [8]. To apply this approach, the developer should specify the goal and define the actions to be performed in case of changes in a constantly monitored environment. Monitoring is implemented as continuous computation of the parameters and conditions for the actions. These conditions are in the regression relationship to ensure robust goal-seeking behaviour. The restriction of the Teleo-Reactive programs is that they require a lot of computations to check the conditions. However, the majority of the conditions are irrelevant to the current situation or might be predicted very precisely. This approach is not suitable when the system resources should be consumed optimally and effectively.

### *C. Modular FT architectures*

Aspect Oriented Programming (AOP) is a promising paradigm intended to improve the modularity of systems by separation of crosscutting concerns. It is achieved by extending the program code behaviour in certain points, without modifying the code itself. This approach is being considered as a useful technique that can support the implementation of HFT. Several examples of applying AOP for supporting modular FT architectures are discussed below.

In [9] the quantitative assessment of exception handling as aspects is provided. Author considers the benefits of using AOP for modularisation of exception detection and exception handling. AOP allows the developer to lexically separate exception handling code from normal application code given that the changes in AOP code will be less intrusive and more simple. However, the limitation of AOP is that there is no possibility to represent global properties of exception control flows. In addition, there are not usable abstractions for composition and reusing of pluggable exception handlers.

Research [10] provides an analysis of the claim that AOP facilitates the modularization of exception handling mechanisms. Authors state that majority of software development methodologies do not give consideration to the design of a system's exceptional behaviour. It is shown that in some cases AOP could even deteriorate the quality of the system. The main result of the study is that AOP will not improve FT in the system with bad architecture. However, it is able to facilitate the structure of well designed systems by separating normal and exceptional activities of the system. Two main contributions of the paper are based on an interplay between AOP and error handling. The former is classification of exception handling code in terms of factors that make influence on aspectisation. The latter is analysis of interactions amongst these factors.

Feasibility and evaluation of using AOP for software implemented hardware FT (SIHFT) is presented in [11]. Authors offer to apply AOP in order to avoid tangling of SIHFT code with code related to the main functionality of the program. Fault coverage and performance penalty were used to assess SIHFT based on aspects. According to the experimental results AOP is convenient for the programs with SIHFT. The authors focus mainly on hardware FT that is implemented in software, however they do not consider FT of the entire system. In addition, this approach does not assume centralised coordination.

Paper [12] estimates the impacts of using AOP and compares AOP with other techniques. The authors measure memory consumption and execution time overhead of the automotive brake controller application after introducing FT mechanisms represented by time redundant execution and control flow checking. These software mechanisms are intended to deal with hardware faults. The implementation is done at a source code level by three approaches: AOP, source code transformation and manual programming in C. Software implemented FT was preferable since it allows the designers to minimise the cost of redundancy by using self-checking and internally fault tolerant electronic control unit (ECU) instead of replicating several ECUs. Authors analysed the pros and cons of the AOP for systematic and application specific implementations. At the function level, FT mechanisms have a very high degree of tangling. This is the reason why AOP introduces significant performance overheads for systematic implementations. However, when knowledge of the application is leveraged, the overheads of using AOP are similar to those caused by manual programming in C, but AOP is more preferable for the developer.

Research experiments evaluating the advantages and disadvantages of explicit exception flows and implicit exception flows using three different exception handling mechanisms based on Java, AspectJ and EJFlow are presented in [13]. This work focuses on a way to facilitate exception handling. The authors state that the goal of exception handling mechanisms is to distinguish normal code and error handling code. However, when the exception related code is modified, the control flow of the program could be unexpectedly affected. Sometimes it is difficult to locate the place where the exception will be handled or where it was raised. The paper claims that the main disadvantage of Java-type languages is that there is no link between the exception rising site and the exception handling site. However, exception control flow is a crosscutting concern and AOP techniques could be applied to facilitate modularity and maintainability in the presence of exceptions. AspectJ provides a way to distinguish normal and error handling code but only syntactically (not semantically). Therefore, the developer does not have any means for specifying how the exception should flow from the rising site to the handling site. In turn, the EJFlow exception handling mechanism introduces two notions: explicit exception channels and pluggable handlers, which are based on AspectJ abstractions. An explicit exception channel abstracts the flow of exception from the rising site to the handling site, whereas a pluggable handler is a special exception handler that could be bound to methods, blocks, classes and packages. Pluggable handlers encapsulate the exception handling code when a predefined point in an explicit exception channel is reached. The experiments showed that textual separation between normal and error handling code does not provide the expected benefits of modular software. At the same time, it was proved that exception channels and pluggable handlers provide more robust and flexible exception handling. Therefore, EJFlow abstractions reduce error likelihood, facilitate software maintainability, improve implementation productivity by providing automated support for developers and make exception control flow more understandable.

## D. Operation Modes

Operation mode (OM) is a special state of the system. OM defines which system functionality will be available at the given instant of time. Or in other words, OM determines the link between the system state and available system functionality, since the capabilities that are available in one mode may not be available in another mode. For example, different set of operations is available for an aircraft during the taxiing, take-off or flight.

In [14] authors promote the notion of mode as partition of the system state space. In addition, they consider the modes as convenient method for modular specification of large state machines. Transition is used to change the mode for the system. Authors provide two possible relationships between modes: serial and parallel. Serial mode means that the system could be only in one mode in one instant of time, whereas parallel relationship assumes that the system is in all of the available parallel modes simultaneously. Serial and parallel modes provide the way to organise assertions about system behaviour in a hierarchical and compartmental organisation. Modes represent control information about behaviour of a system.

Modal systems are presented in [15]. In this study a modal system is defined as an abstract specification of the modes and mode transition. Authors propose formal definitions of the abstractions for specifying modal systems. It is claimed that operation modes are very common in real-time systems, for example a deadline could be dependable on operation mode. Using of modes and modal system refinement facilitates the definition of system properties, transformation of system requirements into model definition and control structure in the system. Event-B was applied to prove mode switching for the state based model.

In this section we considered various approaches to the system structuring and possible ways of the implementation of FT for the systems. Our vision of the HFT architecture is presented in the next section.

## IV. Holistic Fault Tolerance Architecture

In previous sections we considered the issues relating to optimal system operations and the convenience of FT maintainability. To address these problems, we propose the HFT architecture blueprint. This architecture assumes that the application is build out of components whose responsibility is to deliver the main system functionality. The core of this architecture is a special component called the HFT controller, which is supported by several HFT agents. These elements of the architecture ensure dependable and optimal system operation. In addition, they provide a clear view of the system FT mechanisms. The HFT controller coordinates system-wide FT with the assistance of the HFT agents that simplify the implementation and improve the scalability of the HFT controller. Each HFT agent acts as an intermediary between the HFT controller and one or several system components. It is possible to go without the HFT agents for small applications, given that the corresponding functionality will be implemented by the HFT controller.

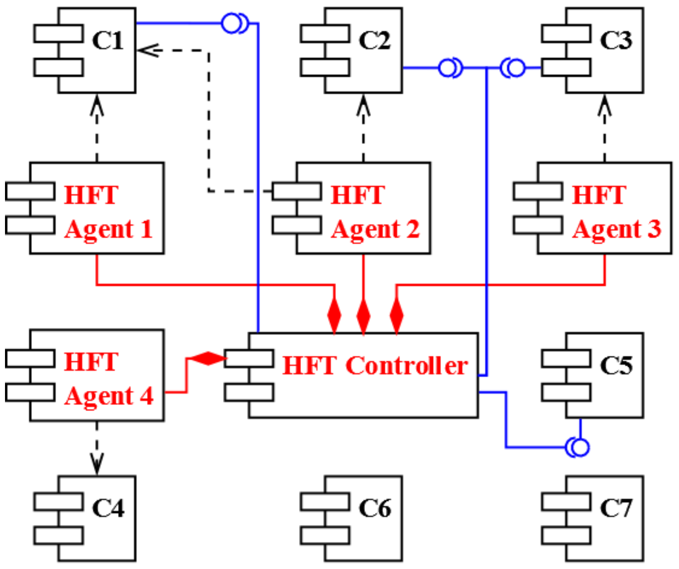

Fig. 1. The HFT architecture

Details of the HFT architecture are provided below.

## A. HFT controller

The HFT controller is a crosscutting unit, which coordinates FT strategies and analyses the performance of the entire system. These tasks are mainly performed with the assistance of the HFT agents, which obtain all required information from the monitored system components and pass it to the HFT controller. Moreover, the HFT controller initiates fault handling and reconfiguration of the entire system after detecting certain erroneous conditions and checking error rates. In this case, apart from the HFT agents' help, the HFT controller utilises public interfaces of crucial system components in order to adjust the reliability, quality of service, performance and energy consumption of the system. However, it should not be aware of the inner structure or encapsulated information of the monitored components because in this case it will be very complex for maintenance and understanding. This is the reason why the knowledge of the HFT controller about the system should be restricted by the general structure of the system and the performance characteristics and average resource utilisation requirements of the system components. The HFT controller should know about the ties between the HFT agents and system components. In all cases when the implementation details of the monitored system components are required to perform some action, the HFT controller should use the HFT agents, which are responsible for their areas of the system and able to provide all required information. Depending on this data, the HFT controller will make system adjustments and reconfigurations.

## B. HFT agent

The HFT agent is a special object monitoring only one or in some cases several system components that are responsible for similar system functionality. It supplies the HFT controller with up-to-date information about the state of the observed components. The HFT agent is not aware of the whole system structure, because its goal is to monitor only parts of the system and pass the obtained information to the HFT controller. The agents have the possibility and right of intervention to the control flow inside the functions of monitored components in order to evaluate the results and handle the errors, since agents are aware of the implementation details of these components.

### C. *Interaction between the HFT controller and the HFT agents*

The HFT controller works with all available HFT agents. In case of error in an observed component, the HFT agent could request the HFT controller for a suitable error recovery or fault handling action. In addition, the HFT agent should detect the erroneous states in the monitored component and propagate this information to the HFT controller. To reduce HFT controller complication, we propose to use discrete enumerations by the HFT controller whenever it is possible. For instance, quality of the operation or result of the function could be presented by the following enumeration: Error, Low Quality, Medium Quality and Good Quality. The additional task of the HFT agent is to perform mapping between real data types and simplified data types suitable for the HFT controller.

### D. *Interaction between the HFT agent and monitored system components*

The HFT agent monitors one or more system components. That includes checking of the results provided by principal functions, performing error detection and error recovery and suppressing exception raising in the monitored components. If the HFT agent monitors more than one component, errors could be detected based on concurrent analysis of two components. In this case, error recovery could affect both components as well. In addition, the HFT agent evaluates current performance and current error rates in its area of responsibility composed of the observed components.

### E. *Interaction between the HFT controller and monitored system component*

The HFT controller has an access to special public interfaces of the system components. These interfaces are used to adjust the component settings after changing of the operation mode or/and to perform fault handling by the reconfiguration of the component. These interfaces give the possibility for the HFT controller to tune reliability and performance of the entire system. We propose to use only public interfaces of the system components for the interaction between the HFT controller and observed system components. All other activities, requiring the amendment of the component behaviour should be done via the HFT agents.

### F. *Fault tolerance functionality*

FT mechanisms in the given architecture are distributed across the entire system, but coordinated centrally by the HFT controller. In some cases, it is worth to introduce some redundancy in FT mechanisms in such a way that the same error could be handled by the component itself and by the HFT agent. The decision on suitable error handling scenarios will be made by the HFT controller depending on the current system state. Such an approach provides the flexibility in the choice of the optimal error recovery scenario. Some errors will be handled by both the component itself and the HFT agent. Only part of the system components needs to be involved in the HFT mechanisms. It makes sense to use only crucial components that globally affect the system operation or could be reconfigured in terms of performance or resource usage. It is definitely more convenient to implement these system components to be “HFT-ready” providing all necessary interfaces and preparing them to work with the HFT controller and HFT agent. However, when the components do not provide such interfaces, for example legacy components, the developers can implement special wrappers or adapters for these components.

### G. *Operation Modes*

Operation mode could be defined as a functional state of the system. Another usage of OM is to provide a non-functional distinction when one mode describes full functionality of the system, whereas another mode is used for exceptional conditions.

With respect to the HFT architecture we offer to apply modes, considering an interplay between reliability, performance and energy consumption. OMs could be applied for the entire system and for separate components in some cases. It is evident that the HFT controller is the most suitable place to control and choose the optimal OM for the system components. Let us consider the following example. Two components are performing some operation. To finish one cycle, the chunk of data should be processed by one component and put in the queue. The second component checks the queue. When there is a chunk of data, it takes this chunk for processing. To provide real-time behaviour, it is necessary to balance the loading of the components. We can specify how to distribute computer resources between these two components and how to balance the data chunk queue. It can be noticed that the HFT controller is a very suitable place to make a decision about resource distribution and OM assignation. HFT agents could monitor the calculation results of these components and supply the HFT controller with this information, so that the HFT controller is able to increase or decrease the quality of service for each component on-the-fly. OM can also be considered as a graceful degradation for the system when some component fails or requires restart. This idea provides the possibility of fault handling with the assistance of the HFT controller, which performs system reconfiguration by choosing suitable modes for the system components.

A system designer should choose which components of the system will participate or will be included in the HFT behaviour and which components will just provide their functionality without being affected by the HFT controller and agents. The system components are classified into four groups.

- Components that are monitored by one or more HFT agent and provide the interface for the HFT controller. Components C1, C2 and C3 in Fig. 1.

- Components that are monitored by the HFT agent/s only and do not provide interfaces for the HFT controller. Component C4 in Fig. 1.
- Components that only provide the interface for the HFT controller. This means that for the given components it does not make sense to observe their inner operation, but they provide good flexibility in tuning performance and resource utilisation. Component C5 in Fig. 1.
- Components that just provide their functionality and are not included in the HFT scheme. Components C6 and C7 in Fig. 1.

In this section we provided the description of the HFT architecture, which includes a special HFT controller and a number of HFT agents. In short, the HFT controller is aware of the general structure of the system and it works with general entities such as quality of the result, error rates and performance, whereas the HFT agent is aware of the implementation details of some part of the application. In addition, we proposed that FT of the system components should be considered in the context of the entire application, but not as a property of these components. The next section provides description of the case study demonstrating practical usage of the HFT architecture for real applications.

## V. Case Study

The case study is centred on the application for the recognition of UK car number plates. The main goal of the application is to demonstrate practical usage of the HFT approach and provide evaluation of the HFT architecture. Thus, it helps to explain the stages of design and implementation of the HFT architecture for the system. The application is not intended to compete with industrial solutions, but it shows how the HFT architecture can be employed during real-world software development processes.

Since our previous work [2], the case study has undergone several changes to assess the benefits of the HFT architecture and to evaluate its scalability and maintainability. Firstly, the previous approach of image processing when the images were uploaded one-by-one was replaced with the queue processing approach. Secondly, we applied AOP to support the modularisation of crosscutting behaviour. Thirdly, the tasks that were performed by the HFT component alone in the first version of the application are now distributed among the HFT controller and HFT agents according to the HFT architecture. This was done to simplify the HFT controller and make it as clear as possible. In the old version the HFT component was presented as a separate class with global knowledge about the system. Sensor and monitoring actions were performed by the HFT component as well. This architecture was suitable for a small application. However, for a bigger application, such an architecture would face scalability issues, since the HFT component should have an access to all important components of the system. The first version of the application was implemented in C# language. The new version is implemented in Java with AspectJ [16] AOP extension.

Five components are responsible for the functional capabilities of the system: Car Number Plate Recognition (CNPR) component, Initial Image Processing (IIP) component, Optical Character Recognition (OCR) component, Number Plates Queue (NPQ) component and Result Checker (RC) component. Simplified structure of the application is depicted in Fig. 2. The CNPR provides the system functionality to the user. This component allows the user to choose a set of images for recognition and monitor the state of the system. In addition, the CNPR shows recognition results for each image, such as time and quality of the operations and the recognised car number plate string. When the user has chosen the images, they are passed to the IIP where the images are processed concurrently.

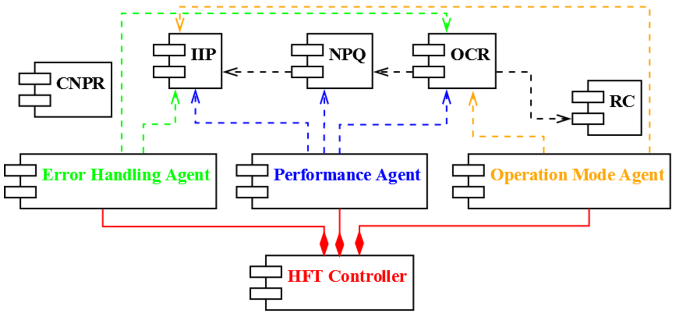

Fig. 2. Architecture of the application

IIP has a choice of two algorithms. The first is light and fast, whereas the second is complex, but more reliable. Depending on the system state, image size and image quality the most suitable algorithm will be chosen for the current image. This component adjusts resolution and contrast of the image, localises the number plate quadrilateral on the image and eliminates the rotation and perspective skew of the number plate cutout. After that the given cutout will be placed onto the NPQ, which stores the cutouts until OCR is ready to process them.

OCR includes three recognition algorithms, which differ in quality and performance. It is chosen dynamically which algorithm to use for the current number plate cutout. In some cases, two or three algorithms may be launched concurrently to provide reliable character recognition. Apart from these three algorithms, the OCR component includes a Character Segmentation (CS) subcomponent, because only one algorithm is able to search the string on the image, whereas the two other algorithms accept only separate symbols. After the recognition process the found string is sent to the RC component, which checks whether the result corresponds to the national format. After this step all information related to the image recognition will be available to the user. The abovementioned sequence of actions describes an ideal case. However, errors could happen at every stage of this sequence. In addition, third-party image processing algorithms were applied and we cannot be sure that they will work without errors or timeouts. This was the reason to introduce redundancy and use several algorithms instead of only one algorithm for each operation. The structure of IIP and OCR components is flexible and algorithms could be added or replaced without significant overheads. To satisfy non-functional requirements for the case study application we implemented the system in accordance with the HFT architecture.

The FT and performance of the system are managed by the HFT controller with the assistance of three HFT agents. The first agent measures the performance of IIP and OCR. In addition, the state of NPQ is monitored. The second agent is responsible for error handling and the third agent supports the operation modes. We applied AOP for the implementation of the HFT agents. *Around* advice [17] is applied to implement custom behaviour before and after invocation of the target method. If necessary, the method result can be substituted. We use around advices for the implementation of performance monitoring and exception/error handling. It should be noted that the erroneous state could be defined not only by catching the exception, but by checking special conditions or values of certain variables as well. For example, low quality in IIP could lead to errors in OCR. If a number of such errors happened, then low quality of initial processing will be considered an erroneous state. However, if character recognition is not affected, then low quality initial processing is considered normal operation.

The HFT controller works only with public interfaces of monitored components, whereas the HFT agents have access to encapsulated information of the system components. HFT agents receive the information from IIP and OCR and transform it to the suitable form for the HFT controller, which works with general entities like quality of the result, type of the error, error rates and execution time of the operations

Several scenarios are possible when an error is detected. The agent that is responsible for error handling will inform the HFT controller about the error in IIP or OCR. Depending on operation mode controller will choose one of the following actions. If the error is not frequent and can be recovered locally then the HFT agent does not intervene in component operation. If the error occurred in OCR when several recognition algorithms were launched and only one algorithm failed then the HFT controller specifies the HFT agent to skip the error, since other functions would return the result. If all recognition algorithms failed then the recognition operation would be stopped and IIP would be started again with complex, but reliable algorithm.

If the HFT controller notices that the error rate of some function is higher that acceptable error rate, it could reconfigure the system component in order to avoid the calls of erroneous function. However, this is possible only in case when the component provides several alternates of the same operation.

The application works with a set of images continuously arriving to IIP and getting processed concurrently. As number plate cutouts are found they are added to the number plate queue, which is monitored by the HFT agent. When the queue is not empty, OCR takes items (number plate cutouts) from the queue for recognition as it performs the recognition of the number plates. The task of the HFT controller is to support the balance of the queue that should not be empty when OCR tries to peek a prepared image. At the same time, the queue should not grow uncontrollably. To implement this task, the HFT controller distributes available computer resources (mainly CPU threads) between IIP and OCR to provide smooth and concurrent execution. When the queue starts to grow, the HFT controller reduces the number of threads for IIP and gives more threads for OCR. However, if the queue becomes empty, the HFT controller implements the reverse action. Moreover, apart from resource allocation, the HFT controller uses the HFT agents to monitor the quality of operation of this components. It could reduce or increase the quality by changing the operation mode for the component.

At the moment two operation modes are available for IIP and OCR, namely reliability and performance. The HFT controller receives monitoring information from the HFT agents and sets the most suitable operation mode through a special public interface of these two components. The mode is chosen depending on current performance and reliability requirements. In some cases, the HFT controller could assign performance mode for IIP and reliability mode for OCR or vice versa if this action will make the system operation closer to the optimal in terms of performance and resource utilisation. The application component (IIP or OCR) should not know about the operation modes and reconfiguration policies, since both could change. Instead, the application component provides the interface for the HFT controller for adjusting its performance, reliability and resource utilisation. This interface does not reveal the inner structure of the component. For example, the following functions are general and do not reveal inner structure: *SetInitialImageProcessingActions* (list of actions), *SetRecognitionAlgorithms* (list of algorithms as parameters), *SetConcurrencyType* (applying algorithms concurrently or consequently), *ChangeErrorRecoveryType* (local, holistic or combined).

AOP can significantly simplify the developing of the HFT agents. In this way, the developer does not need to change the observed components, which almost eliminates the possibility of introducing bugs in existing code. Secondly, we can get access to the private fields and encapsulated information of observed components. These operations should be done with precaution in order to avoid significant alteration of component behaviour.

At the moment we are applying Order Graphs [18] to support modelling and to make the HFT approach suitable for the general case. As part of our future work and further development of the HFT architecture we are planning to undertake the evaluation of the HFT architecture. It will consist of two phases. The first is the definition of quantitative benefits. An application with same functionality will be developed without the HFT approach. After that we will compare performance and resource utilisation requirements on the same input data sets. The second phase is the qualitative evaluation of the HFT architecture from maintainability point of view. We will apply the same two applications (with and without HFT approach) and undertake some changes of FT-related functionality. After that we will compare how many changes are required in both version of the application. For example, how many lines of code changed, how many functions or classes are affected.

The architecture of the case study has been altered since the previous version and now it works with a set of images to clearly demonstrate performance benefits. In addition, we applied AOP to analyse how this methodology can be employed for the implementation of the HFT agents.

## VI. Consclusion

In this paper we presented a Holistic Fault Tolerance architecture, which implies that system-wide FT strategies and resource distribution in the system are coordinated by the HFT controller with the assistance of a number of HFT agents. This architecture and its associated design methods have been applied to a case study application which has passed initial operational experimental validation and analyses. Comparative studies with other approaches will follow in the immediate future.

## References

[1] W. P. Stevens, G. J. Myers and L. L. Constantine, "Structured design," in *IBM Systems Journal*, vol. 13, no. 2, pp. 115-139, 1974.

[2] R. Gensh, A. Romanovsky, A. Yakovlev. 2016. “On structuring holistic fault tolerance,” in *Proceedings of the 15th International Conference on Modularity* (MODULARITY 2016). ACM, New York, NY, USA, 130-133.

[3] T. Anderson, P. A. Lee, Fault tolerance, principles and practice, Prentice/Hall International. 1981.

[4] R. Alexandersson, P. Öhman and M. Ivarson, “Aspect Oriented Software Implemented Node Level Fault Tolerance,” in *Proceedings of the 9th IASTED International Conference on Software Engineering and Applications*, Phoenix, Arizona, USA, 2005.

[5] Microsoft Patterns & Practices Team. (2009). *NET Application Architecture Guide, 2nd Edition*. Microsoft Press.

[6] R. Miller, A. Tripathi. 2004. “The Guardian Model and Primitives for Exception Handling in Distributed Systems”. *IEEE Trans. Softw. Eng*. 30, 12 (December 2004), 1008-1022.

[7] R. Brooks, “A robust layered control system for a mobile robot,” in *IEEE Journal on Robotics and Automation*, vol. 2, no. 1, pp. 14-23, 1986.

[8] N. J. Nilsson, “Teleo-reactive programs for agent control,” *Journal of Artificial Intelligence Research*, vol. 1, no. 1, pp. 139-158, 1993.

[9] N. Cacho, "Supporting Maintainable Exception Handling with Explicit Exception Channels," PhD thesis, Lancaster University, 2009.

[10] F. C. Filho, A. Garcia and C. M. F. Rubira, "Extracting Error Handling to Aspects: A Cookbook," *2007 IEEE International Conference on Software Maintenance*, Paris, 2007, pp. 134-143.

[11] S. Karol, N. A. Rink, B. Gyapjas, J. Castrillon. 2016. “Fault tolerance with aspects: a feasibility study,” in *Proceedings of the 15th International Conference on Modularity* (MODULARITY 2016). ACM, New York, NY, USA, 66-69.

[12] R. Alexandersson, P. Öhman, J. Karlsson, “Aspect-Oriented Implementation of Fault Tolerance: An Assessment of Overhead,” in *Computer Safety, Reliability, and Security*, Springer Berlin Heidelberg, 2010, pp. 466-479.

[13] N. Cacho, F. Dantas, A. Garcia, F. Castor, “Exception Flows Made Explicit: An Exploratory Study,” in *Software Engineering, 2009. SBES '09. XXIII Brazilian Symposium on*, Fortaleza, Ceara, 2009.

[14] F. Jahanian and A. K. Mok. 1994. Modechart: A Specification Language for Real-Time Systems. *IEEE Trans. Softw. Eng*. 20, 12 (December 1994), 933-947.

[15] F. L. Dotti, A. Iliasov, L. Ribeiro, A. Romanovsky. 2009. “Modal systems: Specification, refinement and realisation,” in *Proceedings of the 11th International Conference on Formal Engineering Methods.* Lecture Notes in Computer Science, vol. 5885, Springer, 601–619.

[16] R. Laddad. *AspectJ in Action*. Manning, 2003.

[17] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. Loingtier, J. Irwin, "Aspect-oriented programming," In *Proceedings of the 11th ECOOP*, LNCS 1271, pages 220–242, 1997.

[18] A. Rafiev, F. Xia, A. Iliasov, R. Gensh, A. Aalsaud, A. Romanovsky, A. Yakovlev. (2015). Order Graphs and Cross-layer Parametric Significance-driven Modelling. *15th International Conference on Application of Concurrency to System Design (ACSD'15)*. Brussels.