International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:2, No:4, 2008

An Agent Oriented Approach to Operational Profile Management

Sunitha Ramanujam, Hany El Yamany, and Miriam A. M. Capretz

***Abstract*—**Software reliability, defined as the probability of a software system or application functioning without failure or errors over a defined period of time, has been an important area of research for over three decades. Several research efforts aimed at developing models to improve reliability are currently underway. One of the most popular approaches to software reliability adopted by some of these research efforts involves the use of operational profiles to predict how software applications will be used. Operational profiles are a quantification of usage patterns for a software application. The research presented in this paper investigates an innovative multi-agent framework for automatic creation and management of operational profiles for generic distributed systems after their release into the market. The architecture of the proposed Operational Profile MAS (Multi-Agent System) is presented along with detailed descriptions of the various models arrived at following the analysis and design phases of the proposed system. The operational profile in this paper is extended to comprise seven different profiles. Further, the criticality of operations is defined using a new composed metrics in order to organize the testing process as well as to decrease the time and cost involved in this process. A prototype implementation of the proposed MAS is included as proof-of-concept and the framework is considered as a step towards making distributed systems intelligent and self-managing.

***Keywords*—**Software reliability, Software testing, Metrics, Distributed systems, Multi-agent systems

I. INTRODUCTION

THE Internet era that we are currently a part of has result in a proliferation of data and information which has, in turn, increased the demand for software applications and software systems to handle and manage this data. Software systems have become such an integral part of our world that life without them has become inconceivable. Today, software applications and systems are used in almost all walks of life including industry, education, medicine, business and so on. Millions of users all over the world depend entirely on these applications to conduct their daily activities such as flight booking and bank transactions. Given such a large scale infiltration of software systems into our daily lives and our dependence on the same, any failure or breakdown in these programs would result in substantial financial loss, and, sometimes, even the loss of human lives. Therefore, software reliability engineering is essential in order to improve software operation, thereby saving money and lives.

Authors are with The University of Western Ontario, Dept of Electrical & Computer Engineering, London, Ontario, Canada, N6A 5B9
Telephone: (519) 661- 2111 ext. 85478
Fax: (519) 850-2436 E-mail: mcapretz@eng.uwo.ca

Software reliability refers to the probability of execution without failure for some specified interval of natural units or time [1]. The reliability measurement process is shown in Fig. 1 [2]. The most important step in this process is efficiently constructing an operational profile, which refers to the set of operations or processes for a software application, and the probabilities of occurrence of those operations or processes [3]. Identifying an operational profile and using it to guide testing is an efficient approach because it detects failures, and hence the faults causing them in the order of how often failure occurs [1]. This enables the tester to dedicate more testing time and resources to operations that are most used and that need the most attention. However, as software systems become larger, being composed of thousands of operations and processes, the operational profile defined by Musa [3] may not be an accurate reflection of the real use of the system as depicted by Sommerville [2].

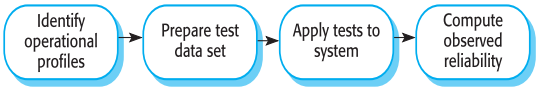


Fig. 1 The reliability measurement process [2]

Further, in distributed systems, software testing presents two fundamentally difficult problems: choosing test cases and evaluating test results. Choosing test cases is challenging because there is an astronomical number of possible test inputs and sequences, yet only a few of those will succeed in revealing a failure. The other problem, evaluation, requires generating an expected result for each test input and comparing this expected result to the actual output of a test run [4].

In addition, current software systems are aiming to be more intelligent and self-managing. This requires more automated and reliable testing in order to keep costs within an acceptable and reasonable range. Yet another challenge involves computing the operations criticality in a distributed system. Operation criticality refers to the importance of an operation in terms of the safety or the value added by satisfactory execution; it also considers the risk to human life, the cost, or the damage resulting from failure [1]. The testing process should be focused on the operations that have high criticality value. This is where the operational profile approach offers significant advantages in terms of identifying the most used operations and faults that have most effect on reliability, and allocating test cases and testing resources in conformance with

the usage of the operations. This leads to a decrease in the time and cost of the testing process by guaranteeing that the essential operations are working well and ensuring that the whole product is efficient. Furthermore, the operations criticality can be used as a metric to organize the testing of the different paths in a distributed system instead of selecting them randomly as in [1].

This paper introduces a novel multi-agent framework to automatically regenerate the operational profile for distributed systems after their release into the market. Agents in the proposed framework are broadly classified into server agents and client agents and their functionalities are described through the roles model and the services model while their communication mechanisms are illustrated through the interaction and acquaintance models. Furthermore, this paper proposes new composed metrics to determine the operations criticality.

The remainder of this paper is structured as follow. Section 2 presents research related to building operational profiles for distributed systems. Section 3 gives an overview of the system objectives and the technology chosen to realize those objectives. The proposed multi-agent framework and the agents' functionalities are described in Section 4 while Section 5 illustrates the functionalities of the agents comprising the system through the various models derived during the system analysis and design phases. An implementation for determining the operations criticality using the proposed MAS is presented in Section 6 and, finally, section 7 summarizes and concludes the paper and provides some future directions.

II. RELATED WORK

The operational profile is a quantitative characterization of how a system will be used and is applied to guide test selection [3]. Developing an operational profile involves progressively narrowing perspective from customers down to operations. The main premise of creating the operational profile is to improve the software reliability for an application through supervision of its testing process. It consists of five steps as follows [3]: Find the customer profile, establish the user profile, define the system-mode profile, determine the functional profile, and finally determine the operational profile itself. Some of the later steps may be unnecessary in a particular application.

Testing driven by the operational profile is efficient, especially in communication software systems, because it identifies failures in order of how often they occur, and hence, the faults causing them. This approach rapidly increases the reliability and reduces the intensity of failures per unit of execution time. However, the performance of this testing technique could be further improved by adjusting many vital factors, such as reducing the number of operations and selecting critical operations in order to schedule the operations testing as well as eliminate any redundancy that may have occurred in that process. Controlling these factors would efficiently increase software reliability and decrease the time and cost of software testing.

Whittaker *et al*. [5] indicate that *a simple distribution of inputs from a human user does not come close to describing the situation.* The operating environment of the software can affect the operation of the software even if the user follows the tested traditional operational profile. The operating system enforces the limits on memory and makes decisions based on the requirements of the other applications in the operating environment [5], [6]. Thus, aspects such as the nature of the data structures, data size and data types are important issues to be considered when executing test cases that deal with the operations and thus have to be taken into account [7], [8]. Therefore, the operational profile must include further information about the operating environment, information about other applications in the operating environment and external data that is used by the application. As a result, an extended operational profile can be built [9].

In addition to the normal operational profile, the extended operational profile includes two additional profiles; the structure profile and the data profile. The structural profile contains both the structure of the system on which the application is running and the configuration of the actual application itself. Data structures can often be characterized by attributes that have numerical value and may change over time. The data profile consists of an application's input values from many users.

Furthermore, the extended operational profile depicts a higher level of reflection than the normal operational profile for any applications in the software market. This extended profile will help organizations validate their systems, and consequently, improve their reliability. However, the selection of test cases is not addressed and also, there is no specified identification for the operation criticality that could help in the testing organization. Moreover, the automatic regeneration of the operational profile is not considered in that work [9].

The greatest challenges that organizations face in validating their applications are those of choosing tests and evaluating test results; this is due to the great variation of test cases and the high cost of the assessment process. The automated model-based testing approach described in [4] could assist in solving those issues. The automated system is designed to support the rapid incremental development of complex distributed systems. It is able to revise the profile, regenerate it, and then run different test suites. In this latter technique, the most important issue involves generating fresh test suites every time the testing is running using discrete event simulation and AI-based meta-techniques. These fresh test cases are more likely to discover newer defects since re-running the same profile-based test suite is inefficient and useless. As high volume automated testing is generated, the output of the system should be monitored. The output checking requires the development and evaluation of expected results, so that, along with fresh test inputs, the system is extended to automatically produce new and expected results to evaluate the test run output. On the other hand, AI-based meta- programming architecture in this approach did not scale well as it imposed a high maintenance cost.

This paper proposes an innovative technique for building a distributed operational profile (DOP) for generic distributed systems using a multi-agent based framework. The DOP will consist of seven steps utilizing the benefits of the normal operational profile in [3] and the extended operational profile in [9]. The first version of DOP will be built statically as suggested in [3], [9]. Consequently, the multi-agent system will automatically modify and regenerate the DOP according to the changes in the distributed system. This regeneration will be done, after releasing the software product in the market; it will be accomplished by monitoring its usage with many clients (i.e. customers) and detecting the changes that might occur at the vendor site due to modification in requirements and or system enhancements.

III. SYSTEM OBJECTIVES and TECHNOLOGY

The primary objective of this research is to automate the process of monitoring customers' software usage in order to establish a connection between the defects and the software usage in the customer environment. Agent technology and software agents were chosen as the design paradigm for the work proposed in this paper due to their ability to augment the human capabilities of reasoning and problem-solving with capabilities of software systems such as multi-tasking and extremely reliable and fast information storage, retrieval, and processing.

A software agent is a piece of software that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors [10]. Changes in a software system and its environment may require changes in the testing strategy. Software agents [10]-[12] are adaptive and can adjust their behaviors based on changes in their environment. They are also autonomous; they can continuously monitor customer's system usage and thus report errors as they are found. The agents that are part of the multi-agent system proposed in this research are also goal-oriented (pro-active), they can generate an operational profile and calculate the required statistics to track the total number of failures as well as the interval between failures. For all of these reasons, we believe that a software agent framework is the best solution for automatically revising and regenerating the operational profile for distributed systems, especially after their release into the market.

In order to achieve the primary objective of this research, there are two major sets of data necessary for collection. The first set is related to the static and dynamic information associated with the customer environment. This includes information on the machine, the operating system, the software configuration, the data held within the database, the movement of data, and the overall operations. This information will assist in revising and regenerating a new DOP that will represent a true reflection of the running distributed system. The second set of information is on the defects that are found by the customers themselves.

As mentioned, the DOP will consist of seven steps; each one will include a different profile. The first five steps are derived from [3] (1. find the customer profile, 2. establish the user profile, 3. define the system-mode profile, 4. determine the functional profile, and finally 5. determine the operational profile itself), the sixth one is comprised of the profiles suggested in [9] (6. the structure profile and the data profile) and the last one describes the influence of the surrounding environment, including the hardware and software. Consequently, the descriptions of the modified profiles are:

1) *Determine the data profile:* A definition of the types or patterns of data and an analysis of its structure.
2) *Determine the machine profile:* A specification of each environment in which the system will run. This profile would help the vendor to track the system behavior in different operating environments.

After the release of the product into the market, the multi-agent based system presented in this paper will start monitoring the product's behavior. The software agents comprising the Operational Profile MAS will be performing four major tasks: monitoring, detection, diagnosis and repair. Monitoring involves observance at both the point of sale and at the customer's location leading to the generation of reports, and conception and/or enhancement of the customer's operational profile. The second task deals with building the agent's capabilities to detect defects by making use of the operational profiles generated by data collected at different customers' sites. Subsequently, after detecting an error, agents work on the diagnosis and attempt to estimate the required actions and tests to fix the error. Eventually, with accumulated knowledge acquired from the environment, agents should be able to not only implement the determined test in the former step but also to automatically repair the found errors.

The next few sections present a detailed conceptualization of the Operational Profile MAS system components with particular emphasis on the functionality, analysis, and design, of each agent in the proposed MAS.

IV. THE MULTI-AGENT FRAMEWORK

As mentioned earlier, the foremost aim of the work proposed in this paper is to rebuild new versions of an operational profile for a distributed system after its release into the market. The new versions of the operational profile should realistically reflect the distributed system as it is running in a given environment. The first version of the operational profile is built statically as mentioned in [3], [9]. This version contains the basic knowledge that will be used to generate the subsequent versions.

Operational criticality is a key input to the generation of operational profiles and is used as a basis upon which to guide testing. It addresses and eliminates two weaknesses of current operational profiles. The first of these is that the efficiency of operational testing decreases as the testing progresses, since more and more parts of the software code have already been examined [13]. Secondly, the random order of executed test cases further reduces the efficiency of testing, because it requires more navigation between different parts of the program [14]. Determining criticality value for the operations of a distributed system helps to decrease the time and cost

involved in testing by focusing on critical operations to ensure effectiveness of the distributed system.

In the work in this paper, composed metrics are used to determine the function criticality. Since, according to [1], particular functions comprise each operation, it will be easier to measure the operations criticality when the functions criticality is known. The proposed metrics include function Complexity (C), Size (S), the Number of Input States (IS) and the Frequency (F) of the function usage. These metrics have been selected based on the criteria mentioned in [15] including the fact that they are quantified, continuous and defined on the basis of the function definition. The description of each metric is as follows:

1) *Complexity:* It evaluates the complexity of an algorithm in a function. A function with a high complexity might be considered a critical function due to the fact that it may contain a greater number of faults [16].
2) *Size:* It can be measured in a variety of ways including the number of all physical lines of code, the number of statements, and the number of blank lines. In this work, we measure the size by the physical lines of code. A function of large size might be considered as a critical function.
3) *Input States:* It is the set of the input values of variables associated with a function and either used by it or affected by it.
4) *Frequency:* This is the number of times that a function is executed during a period of time.

Given the above information, the functionalities of the proposed multi-agent framework can be broadly classified into two categories – server-side functionalities and client-side functionalities. The activities performed by the two categories can be summarized as follows.

A. Server-side activities

1) Interaction with vendors/testers to obtain initial operating conditions and subsequent system changes.
2) Interaction with clients and other servers to gather specific metrics values or to receive information on errors/defects, if any, occurring on the client machines.
3) Determination of appropriate testing plans and error resolution techniques for critical defects.
4) Establishment of a communication scheduling mechanism to appropriately receive and process information coming in from multiple clients.
5) Calculation of criticality values for operations comprising the distributed system in order to re-compute the operational profile if necessary.
6) Generation of new operational profiles based on data from the vendors' servers, the vendors developers, and customer usage.
7) Construction and maintenance of a database to store the various metric values used to determine function criticality, operation criticality values, and newly generated operational profile details.
8) Generation of consolidated reports containing error and resolution information or operational profile details for vendor/tester perusal.

B. Client-side activities

1) Monitoring of client side user log files to gather statistics on usage of functions comprising the distributed system.
2) Calculation of frequency of usage metric values from statistics collected from log files and transmission of the same to the server-side for further processing.
3) Monitoring of client side system for any errors/defects and transmission of the same to the server side for additional processing/resolution.

As stated in earlier sections, software agents have been chosen as an appropriate solution for the automatic and dynamic generation of operational profiles and the analysis and design of the proposed agent-based system is presented in the following section.

V. ANALYSIS and DESIGN of the OPERATIONAL PROFILE MAS

The design model of the Operational Profile MAS has been derived using the GAIA methodology for agent-oriented analysis and design [17]. The output of the GAIA analysis phase results in the Role and Interactions models which are described in the following sections within the context of the proposed MAS.

A. SYSTEM ANALYSIS

System analysis is the process through which developers/designers gain an understanding of the system and its structure at a high level, without focus on the implementation details. During the analysis of the Operational Profile MAS, four major roles were identified. These roles and their functionalities are as follows.

Master Centralized Controller Role (MCC): The Master Centralized Controller is responsible for calculating the criticality values of operations for its servers, collating the criticality values of the other Centralized Controller agents, and generating new operating profiles based on data from vendor servers and customer usage. It registers within its database any changes in system functionalities and any defects encountered by the system, determines the appropriate testing procedures to be performed to resolve the encountered errors and generates a defect report for all encountered errors. It also performs scheduling functionalities to organize the messages received from other Centralized and Client Controllers.

Centralized Controller Role (CC): The Centralized Controller performs a subset of the activities of the Master Centralized Controller role. It calculates the criticality values of operations for its servers and transmits the same to the Master Centralized Controller for consideration during the generation of new operational profiles. It registers within its database any changes in system functionalities and any defects encountered by the clients under its jurisdiction, determines the appropriate testing procedures to be performed to resolve

Role Schema: Master Centralized Controller (MCC)

Description:

The role is responsible for collating and calculating the criticality values of operations for all servers, and generating new operational profiles based on the values. It performs scheduling functionalities to organize messages received from other roles in the system and generates reports (operational profile reports, errors reports) whenever applicable.

Protocols and Activities:

GetInitialOpProfile, GetSystemChanges, TransmitSystemChanges, GetMetricValues, CalculateMetricValues, GetCriticalityValues, CalculateCriticalityValues, RegisterChanges, GenerateNewOpProfile, GetErrorDetails, DetermineTestingType, GenerateConsolidatedReport

Permissions:

Reads *Metric Values* // *from Vendor and ClientControllers*
Criticality Values // *from other CentralizedControllers*

Changes *MCC Database* // *contains changes and defects encountered during distributed system operation*
Operational Profile

Generates *Consolidated Report* // *contains information on errors/defects encountered by system*

Responsibilities:

MasterCentralizedController=(GetInitialOpProfile||GetSystemChange)+.(TransmitSystemChange.(GetMetricValues||CalculateMetricValues).(GetCriticalityValues||CalculateCriticalityValues).RegisterChanges.GenerateNewOpProfile)^{w}||(GetErrorDetails.[DetermineTestingType]*.GenerateConsolidatedReport)*

Role Schema: Centralized Controller (CC)

Description:

The role is responsible for calculating and transmitting (to the MCC) the criticality values of operations for its servers. It performs scheduling functionalities to organize messages received from its ClientControllers and generates reports (operational profile reports, errors reports) whenever applicable.

Protocols and Activities:

GetSystemChanges, CalculateMetricValues, GetMetricValues, CalculateCriticalityValues, RegisterChanges, TransmitCriticalityValues, GetErrorDetails, DetermineTestingType, GenerateErrorReport

Permissions:

Reads *Metric Values* // *from ClientControllers*

Changes *CC Database* // *contains changes and defects encountered during distributed system operation*

Generates *Error/Defect Report* // *contains information on errors/defects encountered by its clients*

Responsibilities:

MasterCentralizedController=GetSystemChange+.((GetMetricValues||CalculateMetricValues).CalculateCriticalityValues.RegisterChanges.TransmitCriticalityValues)^{w}||(GetErrorDetails.[DetermineTestingType]*.GenerateErrorReport)*

Role Schema: Client Controller (ClientC)

Description:

This role monitors the user log files, calculates and transmits the frequency metric. It also monitors the system to identify any errors/defects generated in the system and communicates the same to the CentralizedController role for appropriate handling.

Protocols and Activities:

MonitorLogFiles, CalculateUsageFrequency, TransmitMetricValues, MonitorSystem, TransmitErrorDetails

Permissions:

Reads *User Log Files*

Generates *Usage Frequency Metrics*
Error Details

Responsibilities:

CustomerController=(MonitorLogFiles.CalculateUsageFrequency.TransmitMetricValues)^{w}||(MonitorSystem.TransmitErrorDetails)^{w}

Role Schema: Vendor/Tester

Description:

This role represents the human entity who provides the initial complexity metric values and the initial operational profile. It also transmits any system changes to the Master Centralized Controller and examines operational profile and error reports for appropriate action.

Protocols and Activities:

SpecifyIntialProfile, TransmitSystemChanges, ExamineReport

Permissions:

Read *New Operational Profile*
Consolidated Error Reports

Responsibilities:

Vendor/Tester=SpecifyInitialProfile+.TransmitSystemChanges*||ExamineReport*

Fig. 2 Roles Model for the Operational Profile MAS

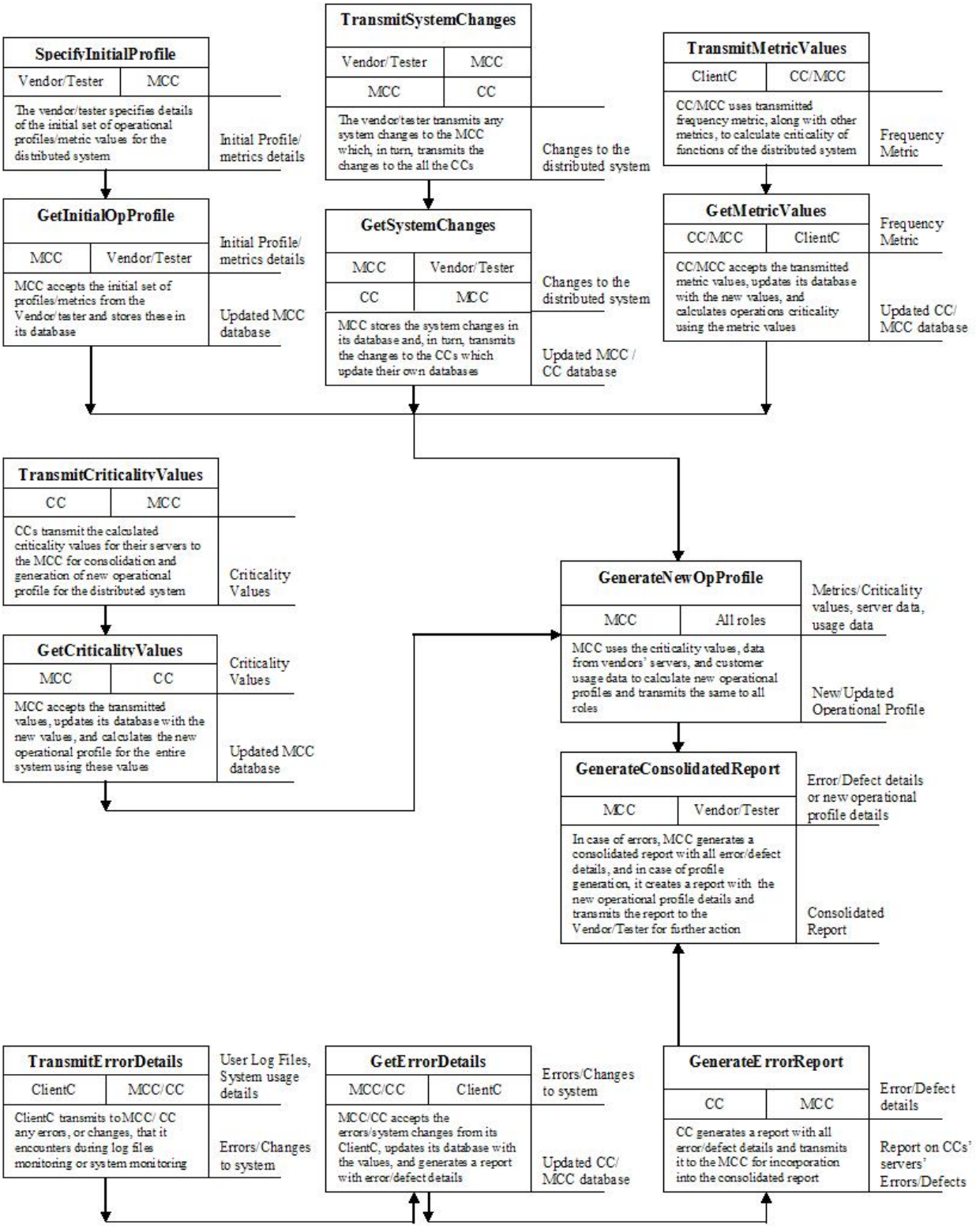
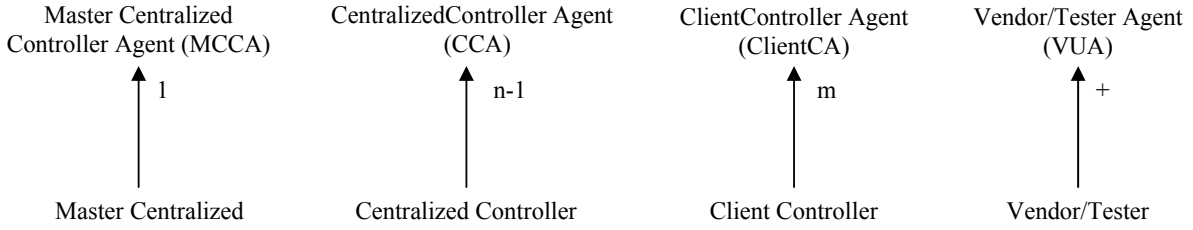


Fig. 3 Interaction Model for the Operational Profile MAS



Where n : Number of servers at the vendors' sites
m : Number of clients at the customers' site
+ : One or more

Fig 4 Agent Model for the Operational Profile MAS

the encountered errors and generates a defect report for all errors encountered within its clients and transmits the same to the Master Centralized Controller for consolidation. It also performs scheduling functionalities to organize the messages received from its Client Controllers.

Client Controller Role (ClientC): This role monitors the user log files in order to gather user usage statistics and, consequently, calculates frequency of usage of each function of the distributed system. It also monitors the system to identify any errors/defects that may be generated during system operation and communicates the same to the Centralized Controller role for appropriate handling.

Vendor/Tester Role: This role represents the human entity who provides the Centralized Controller with the initial complexity metric values to serve as a base for further/future calculations. It also updates the MCC's knowledge-base with any modifications of the distributed system components and is responsible for examining the defect/error reports and determining the appropriate course of action to resolve the same.

The Roles model and the Interaction models developed for the Operational Profile MAS during the GAIA analysis phase are discussed in further detail in the following sub-sections.

1) Roles Model

The key entities (roles) in the system, the permissions associated with them, the activities and protocols they engage in, and the functionalities (responsibilities) of the entities/roles are identified and illustrated in the roles model. The roles identified for the proposed MAS, and their details, are described in Fig. 2.

2) Interaction Model

The dependencies, relationships, and the links between the various roles comprising a MAS are captured through the GAIA Interaction model. The interaction models for the proposed MAS framework are presented below. The purpose of each of the protocol definitions identified for the Operational Profile MAS is listed in Table 1 below while Fig. 3 illustrates the interaction model along with the initiator, responder, input, output, and processing of each protocol.

B. SYSTEM DESIGN

While the analysis phase stays away from the implementation details, the primary focus of the design phase is the transformation of the high-level (abstract) models that were formulated in the analysis phase into lower-level models that can be implemented with ease. The three models created during the GAIA design phase, namely, the Agent Model, the Acquaintance Model, and the Service Model, are described in further detail in the following sections.

1) The Agent Model

The various agent types that comprise the system and the number of agent instances that will be created at run-time for each of the agent types is illustrated in the Agent Model. The agent model for the proposed framework is depicted in Fig. 4. As shown, there is a one-to-one correspondence between the roles identified in the roles model and the agent types.

The Master Centralized Controller is just a Centralized Controller Agent with additional functionalities of coordination and consolidation. In the current version, the Vendor/Tester Agent is assigned the responsibility of designating one of the CCAs as the MCCA. However, in future versions, the CCAs will coordinate and negotiate amongst themselves and choose as the MCCA the most appropriate candidate to perform those additional tasks based on various factors such as load on the CCA servers, processing power available to the CCA servers, number of clients managed by the CCAs, and so on.

Fig. 5 depicts a high-level, conceptual view of the proposed Operational Profile MAS framework in light of the roles and agent types identified in the previous sub-sections.

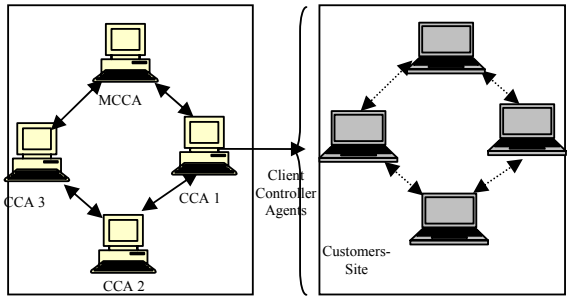


Fig. 5 A Multi-Agent System DOP Management

TABLE I
PURPOSE OF THE PROTOCOL DEFINITIONS

Protocol	Purpose
SpecifyInitialProfile	This protocol enables the human Tester/Vendor to specify the initial Operational Profile (which is used as a starting point) to the MCC role
TransmitSystemChanges	This protocol is used by the human Tester/Vendor to transmit any changes that occur within the distributed system to the MCC role and by the MCC role to propagate those changes to the CC role
GetInitialOpProfile	MCC role uses this protocol to receive, (and update in its database), the initial operational profile details (along with the initial values for the criticality metrics) from the human Tester/Vendor
GetSystemChanges	Any changes to the distributed system components are received from the human Vendor/Tester by the MCC through this protocol
GetMetricValues	This protocol permits MCC and CC to request, from their ClientCs and other CCs (in the case of the MCC), the specific metrics required to recalculate operation criticalities
GetCriticalityValues	This protocol enables MCC to request and receive criticality values for operations on servers managed by the CC role
GenerateNewOpProfile	MCC role uses this protocol to generate a new operational profile based on new criticality values
GetErrorDetails	MCC uses this protocol to receive any errors/defects identified by CC which, in turn, uses this protocol to receive errors/defects identified by the ClientC role
GenerateConsolidatedReport	This protocol is used by MCC to consolidate the reports received from CC into one common report for the entire distributed system. This report is transmitted to the Vendor/Tester for further action
GetSystemChanges	CC uses this protocol to receive any distributed system changes from the MCC role
TransmitCriticalityValues	This protocol is used by CC to transmit the calculated criticality values for operations on its server to the MCC
TransmitMetricValues	This protocol enables the ClientC role to transmit the requested metric values to CC
TransmitErrorDetails	ClientC uses this protocol to transmit any errors/defects it encounters during routine system monitoring to the CC role for further handling.
GenerateErrorReport	This protocol is used by the CC role to generate a report with details on all the errors/defects encountered in the client systems under the CC's jurisdiction as well as in the CC's servers

2) The Acquaintance Model

The acquaintance model highlights the communication links that exist between the various agent types comprising the system and the model for the proposed framework is illustrated in Fig. 6.

3) The Services Model

The last model in the GAIA methodology system design phase, the services model, is used to identify the services associated with each agent type and to describe properties (such as inputs, outputs, pre-conditions, post-conditions, etc.) of the services. The service model for each of the agent types in the proposed framework is given in Table 2.

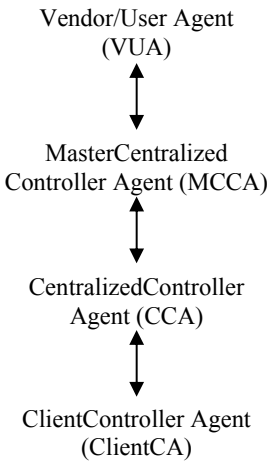


Fig. 6 Acquaintance Model for the Operational Profile MAS

TABLE II
SERVICES MODEL FOR THE OPERATIONAL PROFILE MAS

Vendor/Tester

Service	Inputs	Outputs	Pre-condition	Post-condition
Specify Initial Profile	---	Initial Metric Values, Initial Operational Profile	MAS up and running	---
Transmit System Changes	---	Changes to the Distributed System components	---	Master Centralized Controller database updated
Examine Report	Consolidated Report containing errors/defects encountered or report containing new operational profiles generated	Error resolution procedures (when the report contains Errors information)	---	Procedures for error resolution initiated (in the case of an Error report)

MasterCentralizedController

Service	Inputs	Outputs	Pre-condition	Post-condition
Get Initial OpProfile	Operational Profile from Vendor/Tester	---	---	MCCA database updated with Operational Profile
Get System Changes	Changes to distributed system components from Vendor/Tester	---	Changes have occurred in the system	MCCA database updated and System changes propagated to CCA
Transmit System Changes	---	Changes to distributed system components	Changes have occurred	Criticality values recomputed by CCA
Get Metric Values	Metric Values from other agents	---	---	MCCA database updated with values
Calculate Metric Values	Data from Vendor/Tester, data from system usage/ MCCA knowledge-base	Updated Metric Values	---	MCCA database updated with new values
Get Criticality Values	Criticality Values from CCA	---	---	MCCA database updated with values
Calculate Criticality Values	Metric Values for the four criticality metrics	Criticality values for each function/operation in the system	---	Values used to generate new operational profile
Generate New OpProfile	Criticality Values, data from servers, customer usage data	New/updated Operational Profile	---	MCCA database updated with new operational profile details
Get Error Details	Error details from CCA or MCCA's clients	---	Errors encountered	Error report generated
Register Changes	Error details, system changes, updated metrics/profile	---	MCCA DB up and running	MCCA database updated with current information
Determine Testing Type	Error/Defect details	Suggested Testing procedures	Errors/Defects encountered	Testing suggestions updated in Error report
Generate Consolidated Report	Details of all errors/defects encountered or details of new operational profiles generated by MCCA	Consolidated Report	Errors encountered or new operational profile generated	Error resolution procedures initiated (in case of an Error report)

CentralizedController

Service	Inputs	Outputs	Pre-condition	Post-condition
Get System Changes	Changes to distributed system components from MCCA	---	Changes have occurred in the system	CCA database updated
Get Metric Values	Metric Values from its client agents	---	---	CCA database updated with values

Calculate Metric Values	Data from MCCA, data from system usage/ CCA knowledge-base	Updated Metric Values	---	CCA database updated with new values
Calculate Criticality Values	Metric Values for the four criticality metrics	Criticality values for each function/operation in the CCA servers	---	Criticality values transmitted to MCCA
Transmit Criticality Values	---	Operation criticality values for CCA's servers	---	Values used by MCCA to generate new operational profile
Get Error Details	Error details from ClientCA	---	Errors encountered	Error report generated
Register Changes	Error details, system changes, updated metrics/criticality values	---	CCA DB up and running	CCA database updated with current information
Determine Testing Type	Error/Defect details	Suggested Testing procedures	Errors/Defects encountered	Testing suggestions updated in Error report
Generate Error Report	Details of all errors/ defects encountered in CCA's or ClientCAs' servers	Consolidated Report for CCA's domain of control	Errors encountered	Error report transmitted to MCCA for inclusion in consolidated report

ClientController

Service	Inputs	Outputs	Pre-condition	Post-condition
Monitor Log Files	Users' Log Files	Errors/defects, if any; usage details	System up and running	Function usage details captured
Calculate Usage Frequency	Function usage details	Frequency metric	Usage details available from log files	---
Transmit Metric Value	---	Frequency metric	---	Operation Criticality calculated by MCCA/CCA
Monitor System	---	Errors/defects, if any; changes, if any	System up and running	Any encountered errors/changes transmitted to CCA and, in turn, to MCCA
Transmit Error Details	Log files, system usage	Errors/changes, if any	Errors/changes encountered	MCCA/CCA database updated with changes/errors

VI. IMPLEMENTATION

The chief goal of this implementation is to demonstrate that the suggested composed metrics are efficient enough to compute the operations criticality. A simple financial-based distributed system that belongs to a small company working in the electronics field was studied. Their financial system is composed of many operations; three of them are considered the most important operations. The first operation is creating invoices; each invoice operation is composed of many functions, which include customer data selection, product data retrieval, product price, product stock availability, invoice computation and invoice data storing. The second operation is product manufacturing, whereby a product is built from many parts. This process includes some functions such as parts selection, parts stock availability, parts ordering and product registration. The third operation is creating purchase order for parts. The operation's functions are: supplier's data selection, parts' data selection, parts' price retrieving, producing and storing purchase order data.

An application was implemented to compute the complexity value for each function and produce the criticality degree for each operation in that system. The possible values for criticality in the current implementation are high, medium or low. Fig. 7 is a snapshot of this application. First, the values for the first three metrics (complexity, size and input states) for each previous function were calculated and the metrics' values were stored in the MCCA's/CCA's database. Also, frequency was computed by analyzing the users' log files for this system and stored in the same database. Each user log file includes an Operation Id, a Function Id and the frequency value for the function. Frequency for a function is equal to the average of all users' usage for this function.

Consequently, the CCA/MCCA adds all the metrics' values separately for each function to compute its criticality. The operation criticality value is calculated by taking the average of the criticality values of its own functions. In this work, the range between the maximum and the minimum values of the operations criticality is computed and then divided into three distinct levels to produce the operations criticality degree for all operations in that system. The results show that the three operations mentioned above have a higher criticality degree as compared to other operations in the system. Fig. 8 shows the results.

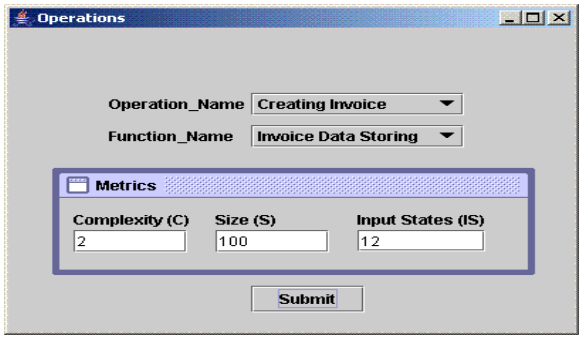


Fig. 7 A Snapshot of Operations' Metrics Calculation

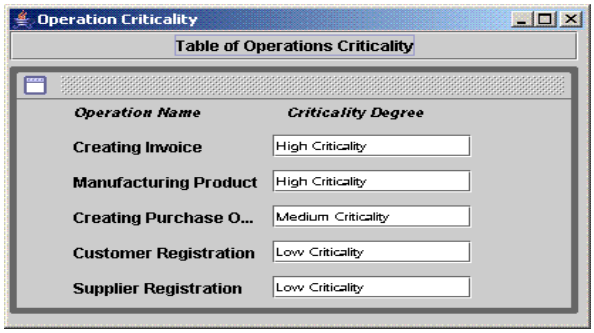


Fig. 8 A Snapshot of Operations' Criticality Results

VII. CONCLUSIONS & FUTURE WORK

An innovative multi-agent framework to automatically regenerate the operational profile for distributed systems after their release into the market has been presented in this paper along with a detailed illustration of various models (namely, roles, interaction, agent, acquaintance, and services model) derived during the analysis and design phases. Agent technology is used as the design paradigm and the proposed framework is comprised of intelligent agents that monitor system changes and user usage at the vendor and client sites in order to efficiently build new versions of the operational profile that represent a more accurate reflection of the distributed system components. This research is aimed at decreasing the time and cost required for testing, thereby increasing the performance and reliability of the distributed system.

Additionally, new composed metrics to determine operations criticality, namely, complexity, size, input states, and frequency, have been proposed in this paper. Using these metrics to determine the criticality value for the distributed system's operations ensures that testing probability is directly proportional to the operation criticality value. This enables testing operations to be focused and organized such that operations are tested according to decreasing order of their criticality values – this guarantees that the testing process will cover all paths of a distributed system.

The proposed framework in this paper is considered a step toward making distributed systems intelligent and self-managing and future work includes incorporation of coordination and negotiation aspects into the CCA agents to enable them to autonomously and intelligently nominate an MCCA amongst themselves. Development of a tool to further validate this model as well as conducting additional testing on other distributed systems to optimize the model's functionality is also under consideration as an extension to the work proposed in this paper.

REFERENCES

[1] J. Musa, Software Reliability Engineering: More Reliable Software Faster and Cheaper, McGraw-Hill. 2004.
[2] I. Sommerville, Software Engineering, Addison-Wesley, 7th Edition, Chapter 24, 2004.
[3] J. Musa, "Operational Profile in Software Reliability Engineering," IEEE Software, Vol. 10, No. 2, Mar. 1993, pp. 14-32.
[4] R. V. Binder, "Automated Testing with an Operational Profile", The Software Tech News, Vol. 8, No. 1. Dec. 2004, pp. 7-10.
[5] J. A. Whittaker and J. Voas, "Toward a more reliable theory of software reliability", IEEE Computer, Vol. 33, No. 12, Dec. 2000, pp. 36-42.
[6] J. Voas, "Will the real operational profile please stand up", IEEE Software, Vol. 17, No. 2, Mar./Apr. 2000, pp. 87-89.
[7] D. M. Woit, "Specifying operational profile for modules". In Proceedings of the ACM International Symposium on Software Testing and Analysis, ACM, 1993.
[8] D. M. Woit, "Operational profile specification, test case generation, and reliability estimation for modules", Technical report, Queen's University, Kingston, Ontario Canada, 1994
[9] M. Gittens, H. Lutfiyya, and M. Bauer, "An Extended Operational Profile Model", In the proceedings of the Fifteenth International Symposium on Software Reliability Engineering, Nov. 2004.
[10] N. R. Jennings, K. Sycara, M. Wooldridge, "A Roadmap of Agent Research and Development," Journal of Autonomous Agents and Multi-Agent Systems, Vol. 1, No. 1, 1998, pp. 5-38.
[11] J. Lind, "Patterns in agent-oriented software engineering," in Proceedings of AOSE Workshop, 2002, pp. 47-58.
[12] M. Wooldridge, "Agent-based software engineering," IEE Proceedings Software Engineering, Vol. 144, 1997, pp. 26-37.
[13] Mitchell, B.; Zeil, S. J.: A Reliability Model Combining Representative and Directed Testing, Technical Report TR 95-18, Old Dominon University, 1995.
[14] M. Grottke and K.D-Zieger, "Systematic vs. Operational Testing: The Necessity for Different Failure Models," in Proc. of the 5th Conference on Quality Engineering in Software Technology, 2001, pp. 59 - 68.
[15] Critical Software Practices for Performance-Based Management: Available: http://www.spmn.com/16CSP.html (URL).
[16] V. R. Basili, and W. L. Melo, "A Validation of Object Oriented Design Metrics as Quality Indicators", IEEE Transactions on Software Engineering, Vol. 22, No. 10, Oct. 1996, pp. 751-761.
[17] M. Wooldridge, N. R. Jennings and D. Kinny, The Gaia Methodology for Agent-Oriented Analysis and Design in Journal of Autonomous Agents and Multi-Agent Systems, Vol. 3, No.3, 2000, pp. 285-312.