Journal of Universal Computer Science, vol. 15, no. 9 (2009), 1871-1885
submitted: 15/8/08, accepted: 25/4/09, appeared: 1/5/09 © J.UCS

A QoS Perspective on Exception Diagnosis in Service-Oriented Computing

Nazaraf Shah
(Department of Computer Science, University of Essex, Colchester, UK
shahn@essex.ac.uk)

Rahat Iqbal
(Department of Computing and the Digital Environment, Coventry University, Coventry, UK
r.iqbal@coventry.ac.uk)

Kashif Iqbal
(School of Computer Sciences, Universiti Sains Malaysia (USM), Penang, Malaysia
kashif@cs.usm.my)

Anne James
(Department of Computing and the Digital Environment, Coventry University, Coventry, UK
a.james@coventry.ac.uk)

Abstract: Unlike object-oriented applications it is difficult to address exceptions in multi-agent systems due to their highly dynamic and autonomous nature. Our previous work has examined exception diagnosis in multi-agent systems based on a heuristic classification method. In this paper, we extend our work by applying an exception diagnosis method to web services (WS) by proposing a unified framework for dealing with exceptions occurring in multi-agent systems as well as in web services. Importantly, we relate the impact of exceptions to Quality of Service (QoS), as exceptions normally degrade the quality of service offered to a service consumer. Our framework consists of a QoS monitoring agent that monitors all interactions taking place between service consumers and service providers. The monitoring agent encodes the knowledge of exceptions, their causes and applies the heuristic classification method for reasoning in order to diagnose underlying causes of monitored exceptions. In this paper, we categorize exceptions into three levels in multi-agent systems: Environment Level Exception; Knowledge Level Exception and Social Level Exception. This paper also discusses different classes of exceptions in web services based on the web service stack.

Keywords: Multi-Agent Systems, QoS, Exception Diagnoses, Heuristic Classfication
Categories: D.2.5, M.4

1 Introduction

Maintaining QoS in the presence of various kinds of exceptions in service oriented systems is a challenging task as such systems are running in a dynamic and open environment where different exceptions (e.g. network connection failure, protocol mismatch, and CPU exceptions) inevitably occur. Such exceptions have implications in achieving the desired level of QoS. In the worst case, these exceptions may result in unavailability of the services, causing the service to renege on its Service Level

Agreement. Therefore, it is important to effectively diagnose the causes of such exceptions so as to initiate effective remedial actions at the right time to eliminate or minimise the adverse effect of these exceptions.

Exceptions can be categorized and diagnosed at three levels in multi-agent systems [Shah, 06]. These levels are: Environment Level Exception; Knowledge Level Exception and Social Level Exception. Environment exceptions normally occur within the internal environment of an agent and its associated software components. Knowledge Level Exceptions are those that result from a wrong selection of action due to the agent's outdated environmental knowledge, or to a misunderstanding of a domain concept. Social Level Exceptions are related to the malfunctioning of an interaction channel, agent dependencies or an organisational relationship.

Exceptions in web services are classified based on a web service stack model [Ort, 09]. They include: Wire Stack Exceptions; Description Stack Exceptions; and Discovery Stack Exceptions. Wire Stack Exceptions are concerned with transport protocols and related technologies. Description Stack Exceptions are concerned with orchestration, composition, service level agreements, business process and the interface description. Discovery Stack Exceptions include exceptions related to technologies for service publication and discovery.

This paper focuses on an exception diagnosis mechanism for a service-oriented architecture. The service-oriented architecture can be realised using multi-agent systems or web services. However, the technologies and mechanisms used to realize these systems are different for web services and agent-oriented services [WS, 09] [Franklin, 96] but they are similar in terms of interoperability, scalability and flexibility.

The main purpose of classifying exceptions, in service-oriented computing, into three classes is to limit the scope of diagnosis mechanism and provide it with a limited search space for exception diagnosis. This helps the diagnosis mechanism to focus its search on relevant classes of exceptions and thus effectively reduces its search space. The exceptions in various classes are organised in a tree structure. The most abstract exceptions are at the top of the hierarchy and the most specialised exceptions are arranged at the bottom of the hierarchy of a tree structure. For further details, readers are referred to [Shah, 09].

Our previous work has examined exception diagnosis in multi-agent systems based on a heuristic classification method [Shah, 05]. In the heuristic classification approach, programs employ an inference structure that systematically relates data to a pre-enumerated set of solutions by abstraction, heuristic association and refinement [Clancy, 85]. It can support diagnosis agents in detecting the root causes of observed faults and in determining the level of such faults.

We have also shown that the issues of QoS from user perspective can be investigated using user-centred design and evaluation approaches including ethnography [Iqbal-a, 06]. Such approaches can help to conduct an effective and rigorous analysis of working practices taking place in real world and real time phenomena. Through user-centred design approaches, we extracted expert knowledge in order to build systems supporting the critical operations of document management systems [Iqbal-b, 06].

In this paper, we extend our work by applying an exception diagnosis method to web services by proposing a unified framework for dealing with exceptions in multi-agent systems as well as in web services. Importantly, we relate the impact of exceptions to Quality of Service (QoS) as exceptions normally degrade the quality of service offered to a service consumer. Our framework consists of a QoS monitoring agent that monitors all interactions taking place between service consumers and service providers. The monitoring agent encodes the knowledge of exceptions, their causes and applies the heuristic classification method of reasoning in order to diagnose underlying causes of monitored exceptions.

The rest of the paper is organised as follows. Section 2 provides the discussion of technological differences between web services and agent services and their interaction styles. Section 3 discusses the proposed framework. Section 4 provides discussion on commitments in relation to QoS maintenance. Section 5 provides the classification of exceptions in agents. Section 6 provides the classification of web services exceptions. Finally section 7 summarises the work and provides an outline for future work.

2 Agent-Oriented Services and Web Services

Web Services (WS) are a technology designed to support interoperable machine to machine interaction over a network [Web, 04]. WS are accessed by standard internet technologies, such as SOAP, XML and HTTP. A web service can be registered (advertised) and inquired about (searched) by using the technology of Universal Description, Discovery and Integration (UDDI) [Clement, 04]. The WS Description Language (WSDL) [Christensen, 01] provides a machine-processable interface in which components that contribute to a composite web service can be executed automatically.

Simple Object Access Protocol (SOAP) [Gudgin, 03] is a text-based (specifically XML-based) communication protocol which can be conveyed by other underlying transmission protocols such as HTTP and SMTP. A SOAP message helps the different executing components of a web service to interoperate when these components are distributed on a network. SOAP is based on the request-response communication style and uses tags to inform a counterpart when faults occur. For example, the tag <faultcode> is used to classify the faults such as *VersionMismatch*, *DataEncodingUnknown*. The tag <faultstring>, in addition, is intended to provide a human readable explanation of the fault. This characteristic is not sufficient when we expect the composite web service to be carried out without human interference. Such a notation makes it difficult for various service components to address the effect of faults on QoS.

Agent-oriented services and web services both realize the vision of Service Oriented Computing (SOC). Agent-oriented services use the agent communication language (ACL) [FIPA, 00], interaction protocols [FIPA, 09] and semantic language in order to manage the issues of interoperability and heterogeneity. Various methodologies have been suggested for developing agent systems [Gomez-Sanz , 04], [Rosaci, 07].

Web services are invoked using basic request-response operations. Composition of existing WS is provided by using the Business Process Execution Language (BPEL) [Das, 07]. The BPEL provides the specifications and standards for defining business processes. It supports basic exception handling by monitoring fault messages and deadlines at each step of the process. The exceptions that are not covered by fault messages cannot be handled adequately by the mechanisms used by the simple request-response model of WS or by BPEL. The BPEL uses a fault handling model that supports nested transactions. Such a fault management model is not suitable for managing QoS.

Software faults in WS may arise at different layering levels, such as protocol level, binding level or SOAP faults. Every layer has its own fault diagnosing and handling mechanism and can handle certain exceptions, or it reports the exception to a higher level in the hierarchy. These exceptions are diagnosed and handled by traditional methods which rely on the underlying communication and programming models. Such methods continue to be developed and can be drawn up by a higher lever process such as that described in this paper [Fetzer, 07], [Tabakow 07].

On the other hand invocations of agent-oriented services are governed by standard interaction protocols. Web services Coordination (WS-Coordination) defines an extensible framework for coordination of component services using a coordinator and a set of coordination protocols [Web, 05]. The standard coordination protocols and mechanisms available for agent-oriented services add complex rich coordination capabilities to these services, whereas coordination mechanisms in WS do not provide such complex and rich coordination capabilities. For this reason we propose the use of agents in order to manage QoS in service oriented applications.

WS fault handling mainly focuses on reporting exceptions in a generic way (for example the SOAP fault) rather then diagnosing underlying causes of these exceptions and enacting resolution strategies in order to maintain the required level of QoS. These approaches rely on exception handling support provided by the underlying programming models. A few research efforts address the issue of exception handling in composite web services [Chafle, 05] [Greiner, 04]. These approaches rely on the BPEL engine to propagate faults to service consumers if not handled by locally available handlers [Web 05] or rules for handling quality of service [Greiner, 04]. These faults, if not handled adequately, will have a severe effect on QoS.

It is essential to have a fault diagnosis and handling and performance monitoring services mechanism in agent-oriented services and web services that provides a uniform QoS related monitoring and handling capability to all services and at the same time provides an element of trust between these services and the QoS monitoring agents. Web services and agent-oriented services are based on open standards for interoperability, such as representation, coordination and interaction. An effective QoS related fault diagnosis mechanism is required to take into account the issues of interoperability and openness. Our proposed fault diagnosis mechanism attempts to address the issues raised above.

3 Proposed Framework

Our proposed approach provides a unified framework for monitoring QoS in service-oriented applications and helps to achieve the desired level of QoS by diagnosing the cases of degradation. In order to offload the burden on services of implementing the complex QoS monitoring and diagnosis capabilities, the proposed approach is realized using QoS monitoring agents. Each service in a service-oriented application is assigned a QoS monitoring agent and communication between service consumer and service provider is carried out via QoS agents. This results in a system composed of QoS monitoring agents and services as shown in Figure 1. Figure 1 shows only one QoS monitoring agent, thus showing centralized configuration of the systems. In a distributed configuration each service has its own associated QoS monitoring agent; such a configuration deals effectively with the scalability and fault tolerant characteristics of distributed systems. It is assumed that the service provider and the service consumer agents are FIPA [20] compliant. The QoS monitoring agent is equipped with the knowledge of various exceptions that have an effect on QoS, their symptoms and their underlying causes. The QoS monitoring agent has both an agent and a web service based interface that enable it to interact both with agent services and web services. All interactions between services take place via a QoS monitoring agent in order to detect any abnormality in QoS. The QoS agreement between service consumers and services are considered as commitments between service providers and service consumers. These commitments are translated in into rules and the rules are then used by the QoS monitoring agent for monitoring QoS.

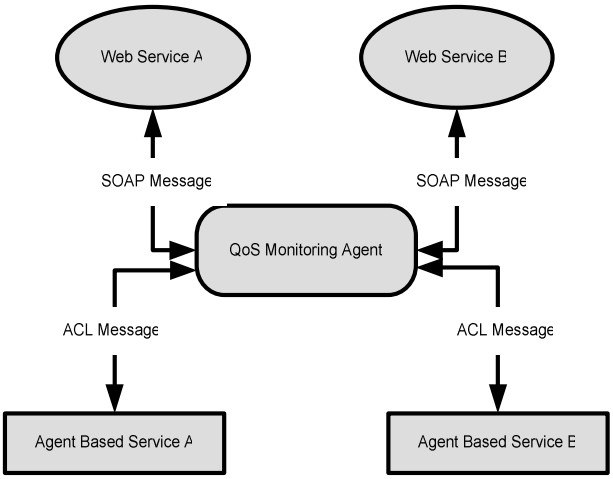

Figure 1: System Structure

The QoS monitoring agent is based on two main components known as the detection module and the diagnostic capability. The detection module is responsible for monitoring the service's interactions and actively seeking symptoms of compromised QoS in interactions. The structure of the monitoring agent is shown in Figure 2.

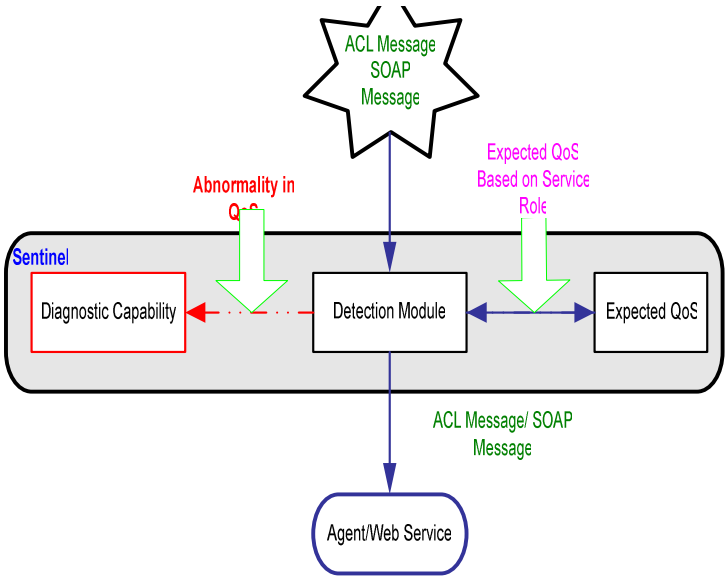

Figure 2: Monitoring and Exception Detection

Figure 2 shows that every message sent to the *Detection Module* is checked for any potential abnormality. The outcome of this module is an ACL/SOAP message or an abnormal event. The expected behaviour is based on the protocol currently being used to govern an interaction and the current state of the commitment. When an abnormality is detected by the detection module an exceptional event is constructed and posted to the diagnostic capability. The *diagnostic capability* applies the Heuristic Classification (HC) [Clancy, 85] approach to uncover the underlying cause of a given symptom. It formulates a diagnostic set to test the conditions that confirm or contradict the presence of underlying causes of the given symptom. The presence and confirmation of such a condition is ascertained by using its plans. The diagnosis plans are activated by posting exceptional events. The invocation of plans simulates the backward chaining reasoning process.

Figure 3 shows the HC process involved when a QoS monitoring agent receives a complaint from its associated service consumer. Four *BeliefSets* are used to simulate the HC method for diagnosing all types of exception. The fifth *BeliefSet* is different, used for complaint related exceptions and other exceptions detected by the *Detection Module*. For example when an ACL/SOAP message containing a complaint is received by a QoS monitoring agent, the complaint information is retrieved from the ACL message and a complaint exceptional event is posted to the *Diagnostic*

Capability. A chain of plans in the *Diagnostic Capability* is invoked in order to diagnose the cause of the complaint. The reasoning process starts by retrieving goals and their associated preconditions from the *Goal BeliefSet* and the *Conditions BeliefSet*. All preconditions of a goal are initialized by asking questions of the associated services. After all preconditions of a goal are initialized, the next step involves the matching of preconditions and the goal with exception rules in the *Rule BeliefSet*. If a rule is matched, an assertion is made in the *Assertion BeliefSet*, otherwise no assertion will be made and the reasoning process will be repeated with the next goal and its associated preconditions. This process continues until a diagnosis agent reaches a conclusion or could not make a conclusion based on its own knowledge and that of its associated agent.

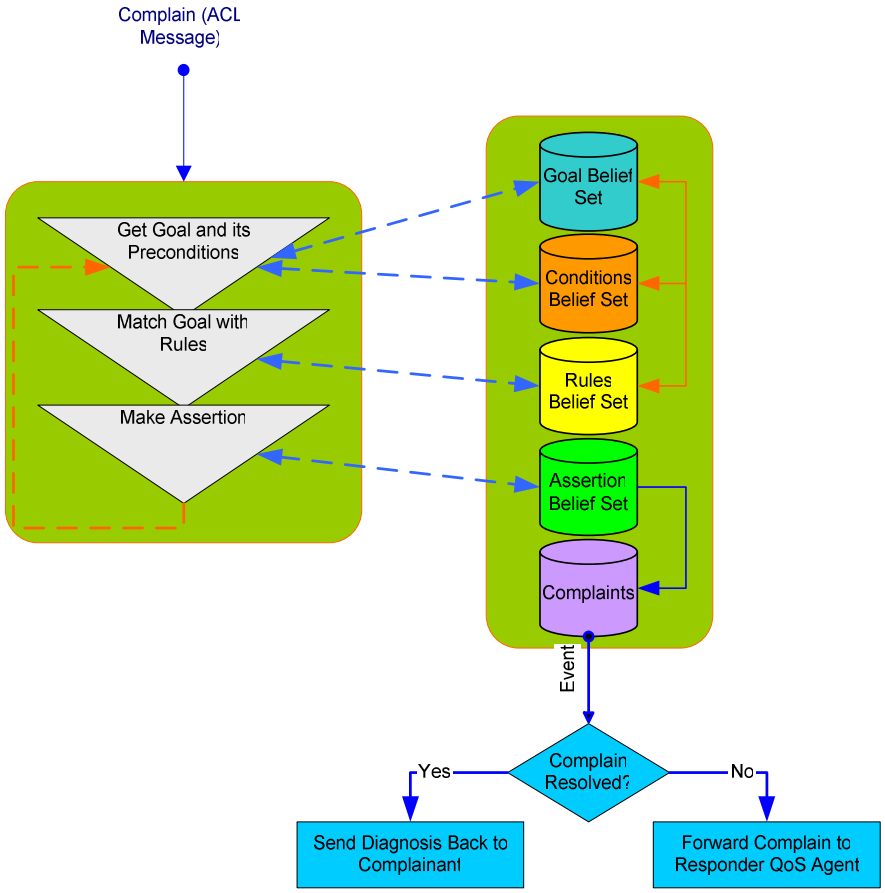

Figure 3: Exception Diagnosis Process

4 Commitments and QoS Aspects

Commitments are formed explicitly by exchanging information regarding an agreement being formed between agents and the conditions on this agreement. Such commitments are known as social commitments. The commitments that a service makes to itself are known as local commitments.

Every social commitment is formed based on a protocol/contract known to the services involved in the commitment. The chosen protocol provides the guidance for the creation, satisfaction and cancellation of a commitment. Service providers and consumers are implicitly committed to the interaction protocols they are employing when forming social commitments and explicitly committed to the performance of a task once an agreement is mutually made by the agents. Social commitments are also influenced by the social policies of an agent based service application and an agent's local policies. A social commitment between service providers "Agent A" and "Agent B" and local commitments are depicted in Figure 4. Social commitment is shown by an arrow emanating from "Agent A" to "Agent B". Local commitments are shown by agents' internal selected intentions. Only social commitments are visible to QoS monitoring agents, local commitments are know to individual agents only. The QoS agents monitor social commitment only, the monitoring of individual commitments is the responsibility their associated service**.** These social commitments have their foundation on QoS related agreements. By using the notion of commitment in agents and realizing the service level agreements in a set of rules, QoS monitoring can be monitored effectively. Broken commitment can be handled by negating and coordinating among various services.

Singh [Singh, 99] treats a commitment as a first class object and defines six different operations on a commitment object known as: *Create, Discharge, Cancel, Release, Delegate*, and *Assign*. We use the *Create*, *Cancel*, *Discharge* operations of a commitment as defined in [Singh, 99] and two of our proposed operations known as *Activate* and *Violate.*

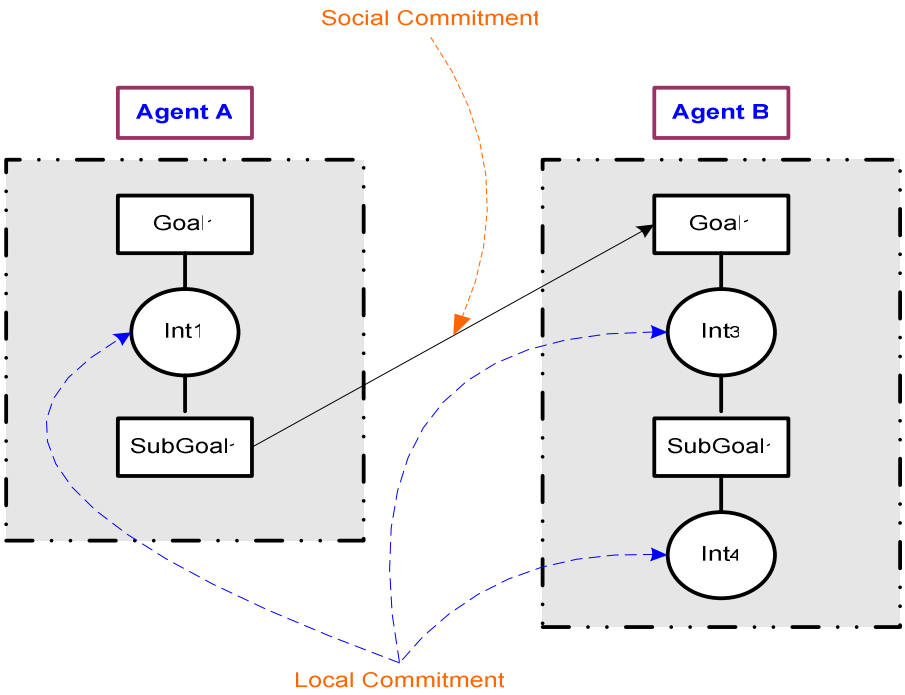

Figure 4: Showing Local and Social Commitment

These operations are performed on a commitment by an agent according to the role of its associated agent.

- *Create:* Commitment is created and put in initial state.
- *Activate:* Commitment status is changed to activated when an agree or an accept-proposal message is received from the commitment debtor.
- *Cancel:* In an open system the conditions for a cancel action must be explicitly stated by the debtor agent, e.g. in the domain of a travel agent a flight ticket cancellation action will refer to the minimum time required for the cancellation action and the penalty involved in cancellation. The creditor must send a valid cancellation message; any message that does not conform to the cancellation conditions set by the debtor is an exception.
- *Discharge:* The debtor agent's diagnosis agent performs the discharge action on the commitment by sending the result of the action back to the creditor agent.
- *Violate:* The debtor agent's diagnosis agent performs the discharge action on the commitment by reporting failure to the creditor agent.

5 Classification of Multi-agent Exceptions

It is proposed that exceptions in service are characterised at three levels, known as: environment level; knowledge level; and social level. Environment and knowledge level exceptions propagate to the social level if not dealt within their corresponding levels. Such propagation compounds the complexity of exception diagnosis. The classification of an exception at different levels provides us with an effective tool to analyse and understand exceptions by limiting their scope to their respective level. The environment level exceptions are concerned with web services and agents whereas knowledge level and social exceptions are concerned with agents only.

5.1 Environment Exceptions

Environment exceptions are those exceptions that occur within the internal environment of an agent or web service and its associated software components. In procedural and object oriented programming models, invalid inputs are considered as environment exceptions. We do not consider such input exceptions as environment exceptions; rather we treat them as social exceptions. The environmental exceptions include, software design defects, garbage data returned from software components, disk full exceptions, I/O exceptions, CPU exceptions or other program exceptions.

Environment exceptions are represented using alphanumeric strings in structured programming environments and by the objects of exception classes in object-oriented environments. Environment exceptions are detected, diagnosed, and resolved by using exception handling techniques provided as a part of the language/environment system. The complexity of diagnosing the causes of environment exceptions increases with the increase in number of software components being used by an agent/web service for supporting its functionality. When an agent's/web service component fails to detect, diagnose and deal with an environment exception, the exception then propagates to the agent/web service level environment. The subsequent diagnosis of this exception is made harder by the potential environmental differences. This is due to the fact that different programming environments use different exception handling models and their exception representations also differ greatly.

5.2 Knowledge Level Exceptions

The knowledge level (KL) is defined by Newell [Newell, 82] as a computer level that sits above the symbol level. The KL is characterised by knowledge as the medium and the principle of rationality as its law of behaviour. The principle of rationality [Newell, 82] states that if an agent has knowledge that one of its actions will achieve one of its goal, and then it will take that action. The KL enables us to view an agent in terms of its actions and goals, together with the principle of rationality, without getting bogged down with the agent's internal structure. The selection of an action for achieving a goal also depends on the agent's current knowledge about its environment. An agent may make the wrong action selection if its assumptions about its environment are not valid.

The KL is an abstraction made by an observer. The diagnosis agent acts as an observer associated with a problem solving agent. Being able to play the role of an

observer, the diagnosis agent has the knowledge of goals and actions of its associated agent. The KL exceptions are those exceptions that result from a wrong selection of action due to the agent's outdated environment knowledge, or to a misunderstanding of a domain concept. The diagnosis agents detect and diagnose the exceptions that are related to wrong action selection for the achievement of a goal. The domain related concept exceptions are dealt with using the ontology of the domain concepts and the actions allowed on those concepts.

Agents are knowledgeable entities and interactions between them are considered in terms of knowledge exchange [Gaspari, 98]. The knowledge of an agent in a given society can be divided into two types: what is known as self knowledge (action, goal); and knowledge about its environment, including other agents. The knowledge about their capabilities and how to interact with other agents is a level above the KL and is known as social knowledge.

5.3 Social Level Exceptions

The focus of the KL is on a single asocial agent. When we consider a society of agents, we need to consider social aspects involved in the effective management of the society. The KL says nothing about the social aspects of a mulit-agent system (MAS). A new level is needed to accommodate the social character of an agent society. Jennings proposes such a new level above the KL, known as the "social level" [Jennings, 00]. It is concerned with social level principles for MAS's.

The components of the social level are agents, interaction channels, dependencies, and organisational relationships. The behaviour law at the social level is the principal of organisational rationality instead of the principal of individual rationality.

Exceptions related to the malfunctioning of an interaction channel, agent dependencies or, organisational relationship, are classified as social exceptions. The majority of social level exceptions are context dependent [Shah, 05]. The diagnosis agent contains the knowledge of the way the social relationships are established, maintained and discharged in a community of autonomous agents. Our proposed mechanism assumes a peer-to-peer organisational relationship among problem solving agents, which is conforming to FIPA standards for an open MAS.

A diagnosis agent starts the diagnosis process when an exception surfaces at the social level and moves down to the knowledge and environment levels in order to classify the exception and diagnose its underlying cause. If the exception belongs to the social level, then the diagnosis agent will not investigate the knowledge level or environment level. For example lost message or dropped commitment (in favour of a better choice or due to malice) exceptions are social level exceptions and do not involve the knowledge level or the environment level diagnoses. On other hand when a protocol state skip exception surfaces at the social level, its underlying cause could be that agent is deliberately trying to fool the other agent or it may be an environmental level exception due to a bug in the agent code.

The exception diagnosis process employed by a diagnosis agent is depicted in figure 5.

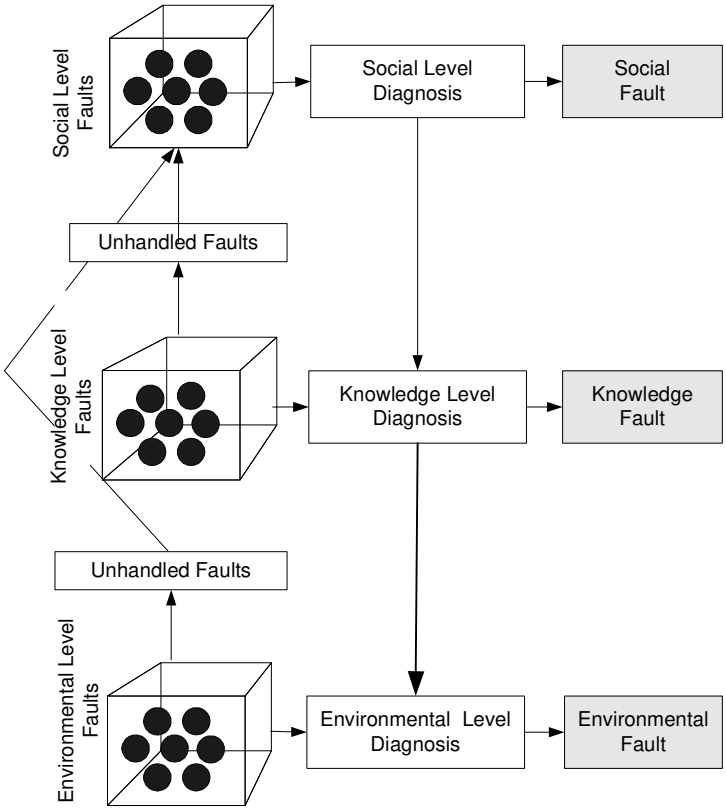

Figure 5: Levels of Exceptions

6 Exception Classification in Web Services

Although agent-oriented systems and web services share same types of low level exceptions, their higher level exceptions differ due to difference in their computational model and their associated attributes. The composition of web services into business processes may introduce or lead to process related exceptions, which may not manifest themselves as low level computational exceptions. Diagnosing the cause of such exceptions involves the use of service level agreements and policy driven exception management. We classify web services exceptions based on the web services stack model and a class of exception that may result from composition of web services in a business process.

6.1 Wire Stack Exceptions

The exceptions in the wire stack are a class of exceptions that comes from technologies and protocols that are used to realise the wire stack. This stack is basically concerned with the actual exchange of data, i.e. protocols and technologies

associated with physical exchange of data. Exceptions related to SOAP, MIME, HTTP, TCP etc belong to this class of exceptions.

6.2 Description Stack Exceptions

This class of exception is concerned with orchestration, composition, service level agreement (SLA), business process, interface description and policy related exceptions. For example hardware failure, communication failure, hardware and software upgrade are mentioned as exceptions in SLA. Such exceptions are exceptions to rules specified in SLA. Business process exceptions are high level exceptions of varying origin. Such exceptions require a elaborated diagnosis mechanism in order to discover underlying cause of the manifested exception

6.3 Discovery Stack Exceptions

The discovery stack is concerned with technologies for service publication and discovery. These technologies conform to principle of interoperability. All exceptions that may occur in inspection, publication and discovery sub-layers are known as discovery stack exceptions. This class mainly includes WSDL faults and these faults are identified by the name of fault and target name space of the corresponding port type.

The main purpose of classification of web services exception into three classes is to limit the scope of the diagnosis mechanism and limit its exception diagnosis search space. The security, management and QoS apply to all components of the web service as described above. Exceptions occurring for any of above components or violation of security have effect on QoS of the system.

7 Conclusions

We have presented a QoS monitoring and diagnosis approach for agent-oriented services and web services. The proposed approach monitors and diagnoses the underlying causes of commitment violation heuristically and interactively without violating the autonomy of the services involved. The proposed architecture is FIPA/web services compliant and can be integrated with FIPA compliant agent-oriented services and web services. The proposed approach is based on well known and well accepted technologies, such as heuristic classification, coordination protocols, state machines, the FIPA ACL and web services standards, which makes it suitable for providing QoS monitoring of services to any FIPA compliant, agent-oriented services system or web services system.

In this paper, we have categorized exceptions into three levels in multi-agent systems: environment level exception; knowledge level exception and social level exception. We have also discussed different classes of exceptions in web services based on the web service stack. These include: wire stack exceptions; description stack exceptions; and discovery stack exceptions. The main purpose of classifying exceptions, in both multi-agent system and web services, into three classes is to limit

the scope of the diagnosis mechanism for a particular exception and thus limit its exception diagnosis search space.

Our future work will include accumulating more knowledge related to exceptions and extending the exception tree. Following that we intend to implement various remedial strategies for dealing with different exceptions both in multi-agent system and web services. We will also evaluate this work by applying the approach to the supply chain management domain.

References

[Chafle, 05] Chafle, G., Chandra, S., Kankar, P., Mann, V.: Handling Faults in Decentralized Orchestration of Composite Web Services, In Proc. ICSOC 2005, LNCS 3826, pp. 410-423, 2005

[Christensen, 01] Christensen, E., Curbera, F., Meredith, G., Weerawarana S.: Web Services Description Language (WSDL) 1.1, W3C, 2001, http://www.w3.org/TR/2001/NOTE-wsdl-20010315, 15th March 2001

[Clancy, 85] Clancy, W., J.: Heuristic Classification, Artificial Intelligence, 27, Elsevier, 289-350, 1985

[Clement, 04] Clement, L., Hately, A., Riegen, C., Rogers, T., UDDI Version 3.0.2, OASIS, http://uddi.org/pubs/uddi-v3.0.2-20041019.htm. 2004

[Das, 07] Das, M., Yiu, A., Kund, K.,: A Close Look at BPEL 2.00, Sys-Con Media, 2007, http://www.sys-con.com.node/434430

[Fetzer C 07] Fetzer, C., Felber, P., Improving Program Correctness with Atomic Exception Handling, Journal of Universal Computer Science, 13, 8, 2007

[FIPA, 00] FIPA Communicative Act Library Specification, Available: http://www.fipa.org/specs/fipa00037/SC00037J.pdf, 2000

[FIPA, 09] FIPA Interaction Protocols Specifications, Available: http://www.fipa.org/repository/ips.php3. Accessed on May 2009

[Foundation, 09]Foundation for Intelligent Physical Agents, Available: http://www.fipa.org/. Accessed on May 2009

[Franklin, 96] Franklin, S., Graesser, A. :Is it an Agent, or Just a Program?: A Taxonomy for Autonomous Agents, In Proc of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages, 21-35, 1996

[Gaspari, 98] Gaspari, M.: Concurrency and Knowledge Level Communication in Agent Languages, Artificial Intelligence, 105(1-2), Elsevier, 1-45. 1998

[Gomez-Sanz, 04] Gomez-Sanz, J., Pavon, J., Methodologies for Developing Multi-Agent Systems, Journal of Universal Computer Science, 10, 4, 2004

[Greiner, 04] Greiner, U., Ram, E.: Quality-Oriented Handling of Exceptions in Web-Service Based Cooperative Processes", In Proc of EAI-Workshop 2004 - Enterprise Application Integration, Oldenburg, Berlin, 11-18, 2004

[Gudgin, 03] Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J-J., Nielsen, H., F.: SOAP Version 1.2 Part 1: Messaging Framework, W3C, 2003: http://www.w3.org/TR/2003/REC-

soap12-part1-20030624/

[Iqbal-a, 06] Iqbal, R., Shah, N.H., James, A., Younas, M., Chao, K-M.: A User Perspective of QoS for Ubiquitous Collaborating Systems", In Proc of 10th Intl Conf. on Computer Supported Cooperative Work in Design Conference (CSCWD 2006),IEEE Computer Society Press, 2006

[Iqbal-b, 06] Iqbal R., Shah N, James A., Younas M., Chao K-M "Developing Ubiquitous Collaborating Multi-Agent Systems Based on QoS Requirements", Computer Supported Cooperative Work, Springer-Verlag, Lecture Notes in Computer Science, 2006

[Jennings, 00] Jennings, N., R.: On Agent-Based Software Engineering", Artificial Intelligence, Elsevier 1179, 277-297, 2000

[Newell, 82] Newell, A.: The Knowledge Level, Artificial Intelligence, 19, 87-127 Elsevier, 1982

[Ort, 09] Ort, E.: Service-Oriented Architecture and Web Services - Concept, Technologies, and Tools. Available: http://java.sun.com/developer/technicalArticles/WebServices/soa2/soa2.pdf. Accessed on May 2009

[Rosaci 05] Rosaci, D, Exploiting Agent Ontologies in B2C Virtual Marketplaces, Journal of Universal Computer Science, 11, 6, 2005

[Shah, 06] Shah, N., Lo, C-C., Huang, C-L, Chao, K-M., Godwin, N.: Exception Diagnosis for Agent-Oriented Services, Systems, Man and Cybernetics, 3017- 3022, 2006

[Shah, 09] Shah, N., Iqbal, R., James, A., Iqbal, K.: Exception representation and management in open multi- agent systems, Journal of Information Sciences, 2009.

[Shah, 05] Shah, N., Chao, K-M., Godwin, N., James, A.: Exception Diagnosis in Multi-Agent Systems, In Proc of IEEE/WIC/ACM International Conference on Intelligent Agent Technology, 483-486, 2005

[Singh, 99] Singh M., P.: An Ontology for Commitments in Multiagent Systems: Toward a Unification of Normative Concepts, Artificial Intelligence and Law, volume 7, 97-113, 1999.

[Tabakow 07] Tabakow, I., Using Place Invariants and Test Point Placement to Isolate Faults in Discrete Event Systems, Journal of Universal Computer Science, 13, 2, 2007

[Web, 04] Web Services Architecture, 2004, Available via: http://www.w3.org/TR/ws-arch/

[Web, 05] Web Services Cordination (WS-Coordination), 2005, Available: ftp://www6.software.ibm.com/software/developer/library/WS-Coordination.pdf

[WS, 09] WS-I Organization, Available via: http://www.ws-i.org/. Accessed on May 09