DSoS

IST-1999-11585

Dependable Systems of Systems

Addressing Dependability in Multiple Domains of Management

By

Panayiotis Periorellis

Report Version: Deliverable (DCS3)

Report Preparation Date: 1 December 2002

Classification: Public Circulation

Contract Start Date: 1 April 2000 **Duration:** 36m

Project Co-ordinator: Newcastle University

Partners: DERA, Malvern – UK; INRIA – France; CNRS-LAAS – France; TU Wien – Austria; Universität Ulm – Germany; LRI Paris-Sud - France

Project funded by the European Community under the "Information Society Technology" Programme (1998-2002)

Addressing Dependability in Multiple Domains of Management .....

Abstract

The purpose of this deliverable is to present our final results from the Travel Agent case study and through that investigate integration issues amongst computer systems that belong to different organizations, are placed behind strong organizational boundaries and handled within multiple domains of management. The conclusions have been drawn mainly from the Travel Agent (TA) case study although some examples have been drawn from the EXaMINE case study which we studied in the later stages of the DSoS project. The document is outlined as follows. We are presenting and discussing a number of results that we received from the case studies in the last few months and since our last deliverable [PCE4]. We will show that in order to address the issues raised by the case studies we need to carefully think about the architecting of a Systems of Systems (SoS). Note here that we are interested not merely in the architecture but in the process of architecting i.e. the rationale behind providing an adequate architecture that addresses the issues surrounding systems of systems i.e. autonomous systems, that are integrated while maintaining their organizational boundaries.

1. Introduction

What we have learned so far is that building systems of systems is not by any means a only technical challenge. In fact many of the major problems in the TA case study and most of the problems in the EXaMINE case study stemmed from organizational issues.

The notion of a SoS challenges one aspect of software engineering that we have so far been taking for granted. Traditionally since information systems existed within an organizational domain they reflected the strategy, market, political and economic culture and history of the organization which they belonged. Since all these aspects stem from the same source (the organizational or in the case of EXaMINE the nation) the emphasis was put into how to simulate and incorporate these aspects into an operating system. Change or evolution has always been a difficult issue to deal with in technical terms although vital in the survival of systems. Open systems architectures and frameworks serve well as vehicles for dealing with this issue. The SoS however challenges the main

assumption. Since we put together autonomous systems i.e. systems with existing strategic, tactical and operational goals, political and economic culture and history, issues such as common interest or common goals (if they exist) have to be made explicit.

So where are these issues made explicit?

Our interest is mainly in dependability and we are therefore trying to provide a framework for putting together systems in a dependable manner. The notion of SoS however challenges this issue too. Another assumption we have also been taking for granted is that of a universal (or system wide) judge. A Judge which identifies faults, errors and deviation of intended service. This notion of judgment exists within the same organizational boundary as the information system and is exercised by the organization itself. When we put together systems of systems we also bring together these judging systems in a network of goals mapped onto computer systems, assessed by autonomous judgment systems. Bearing in mind that we have such a diversity of goals and "intended behavior"

Who can judge the "intended behavior" of systems of systems and furthermore who is the judge of the System of Systems?

In this deliverable (having already studied 2 case studies) we have identified that most problems or failure modes can be classified into composition, definition of the emerging service, communications and evolution and we therefore present these in turn in sections 1,2 and 3. We also however acknowledge the fact that a system of systems, the very notion itself creates a new problem space which needs to be explored. At this point of time we lack the conceptual tools that would allow us to study this problem space. We therefore need a framework that addresses all issues we have discussed in this and other deliverables so far. In the sections 4 and 5 of the document we attempt to provide some concepts which we feel are useful for enabling SoS developers to think and investigate such SoS related issues.

We show the need to investigate this 'new' problem space (i.e. integration between systems that cross organizational boundaries) using abstract concepts in terms of roles and responsibilities [PJD2002] as well as the relations between them which we analyze in terms of conversation theory [MD1999]. Following a number of observations about these analyses techniques, the document concludes with a discussion of future research in this area.

2. Architecting a System of Systems

The interesting dimension that DSoS adds to large systems is that of multiple ownership which in cases implies distributed control. Current techniques such as UML do not address this issue or rather do not address this issue in the detail required. If control is distributed between the various sub-systems then we need to examine two things. Firstly the relation between subsystems and the responsibilities these sub-systems hold. Are they responsible for the benefit of each other? Are they sharing the same benefit by being related as part of a SoS? Can they share resources?

Systems where multiple domains of management and control are involved belong to a particular class of systems and should be viewed from a different perspective. Traditional analysis techniques and solutions do not always apply. The fact that some modeling techniques have been successful in the sense that they provide a rich notation or a sound process for analyzing and describing systems does not imply that they are fit to be used everywhere.

For the past 3 years we have dealt with a class of systems that required a lot more effort and collaboration between various disciplines in order to understand the extent of the problem and allow us to provide pragmatic solutions that address problems of a wider spectrum. We are moving towards developing re-usable, mobile pieces of code that can be used over the web via SOAP or over IIOP while at the same time maintaining their sense of autonomy.

Pieces of software that work as part of a larger structure can expose a number of interfaces that would allow them to be used by other structures as part of other systems which are again completely autonomous pursuing their own goals and objectives. We can actually implement software that can be successfully used as part of two distinct systems which are even based on different architectures.

Some of the architectures [CORBA][JAVA RMI] employed for distributed components leave the code that is to become a distributed object completely untouched or unspoiled as some would say. Since the majority of the code is built as a wrapper then it is apparent that a piece of software can indeed be part of several architectures, responding to requests coming over SOAP or IIOP as well as calls made locally.

When we talk about systems of distributed control then we ought to discuss responsibilities. Not responsibilities in the sense that class *a* has to perform a certain number of methods but responsibility in the broader sense of the word. Such that sub-system *y* is responsible for carrying out process *x* successfully in order to maintain the state of affairs. Sub-system *y* is liable for performing *x*. If fault *z* is exposed to the world via the wider system, the world should also hold sub-system *y* liable only and not the wider system. This poses a number of challenges. First of all one may ask how do we make systems responsible and how do we identify responsibilities that can be held by each system? Furthermore how can we inform the world of the responsibilities held by each system when some of them may indeed be completely transparent. Existing notations such as UML or MOT do not address this issue. They are primarily concerned with computer systems where ownership is centralized. This does not imply by any means that processing is centralized but that there is a single domain of management controlling the entire system which may or may not be widely distributed. In our view this creates a new problem space that needs to be explored and the reason we want to explore this space is to be able to reason for an architectural style that addresses the issues raised in this document. Since we are viewing systems in a different light we cannot assume design patterns that have been successful in the past. We need to question design decisions and assumptions that experienced software engineers would take for

granted. Computer systems reflect organizational purpose, objectives, goals. In the case of system of systems they also reflect contractual agreements between participating components. There is a large body of literature on how to build computer systems that reflect organizational objectives but we lack literature, tools and general engineering concepts that allow us to build systems that reflect co-operation between organizations and therefore simulate these contractual agreements. Therefore we need to view computer systems that participate in a larger SoS in terms of the role they play, responsibilities they hold and duties they need to perform in order to maintain the state of affairs. In the following section we take a look at some of the final results and observations drawn from our case studies before we discuss a more generic framework for exploring some of the issues (composition, evolution, transaction) this new problem space sets.

3. Case Study Results and Observations

In this part of the document we are addressing a number of issues that were raised during the later development of the TA [DMS3] study and also presenting some observations from EXaMINE [EXaMINE1]. All the problems that we are discussing here stem from the way the 'emerging service' is composed, regulated, operates and evolves bearing in mind that all these aspects are dependent on the individual autonomous systems that are coordinated to provide the emerging service. We discuss each on them in turn.

Composition

We view composition of the "emerging service" as the process that ensures that goals embedded in the design and execution of individual autonomous systems does not bring the services of the SoS into conflict. We consider this view appropriate since the composition of the emerging service is based on systems of diverse objectives, operational policies, architectural styles and implementation. In other cases (EXaMINE) these conflicts may take the form of political or social disputes. Having already reported on the issues regarding the crossing of organisational boundaries and fault tolerant policies in SoS [PCE4], we need to look how goals embedded into an autonomous system can affect the composition of the SoS. In particular we want to identify, eliminate, and reconcile conflicting goals while at the same time promote complementary ones. Note

here that we use the term goal in order to abstract from both case studies since in the TA what we regard as goals maps onto organisational strategy and in the case of EXaMINE maps onto national interest. In both cases however is important to acknowledge the need for composing emerging services according to a rationale. Bear in mind that the purpose of a system of systems is to compose an 'emerging service' by making use of existing services offered by autonomous systems, each of which has its own goals, culture, history etc.This need for rationalizing on certain configurations of the emerging service became apparent during the structuring of the TA case study and in particular while considering the levels of exception handling.

We have showed already [RPZ2003] that in distributed systems of systems there are several layers of exception handling. Each layer addresses a particular level of the system namely the servers, clients and middleware. There is however another layer of exceptions that do not manifest themselves as typical faults to be captured by the application or the compiler or the runtime environment (regardless of the consequences). We are referring to exceptions that manifest themselves as data inconsistencies, data inadequacies or data mismatches.

In particular in the cases where data obtained via the different autonomous booking systems could not be composed into an item because the data was addressing a different type of client, the policies regarding cancellation were inconsistent, or at times, dates of the items on offer did not match. Similarly, in the context of the EXAMINE case study [EXAMINE1] composing an EU-wide EPS (electric power system) would require an overall control to be exercised by some authority governing body responsible for the entire system. In this example the policy mismatches we mentioned in the TA would manifest themselves as political conflicts since national interests prevail over EU law.

Going back to computer based system of systems, in order to identify these types of exceptions we need to perform additional handling as well as enrich the conditions under which data is composed into an emerging service. Furthermore the need for intelligent and dependable composition becomes apparent when we consider different types of

clients. A web client for example may serve well for providing the user with a variety of items and involving him in the selection process. A WAP (wireless application protocol) client however is a lot more restricted and requires more precise composition. It has already been discussed [AL2001] that goal-orientated approaches to system composition can increase the visibility of goal-relationships that can exist. This is an important benefit of such a compositional approach as such goal-relationships will be subtly specific to the particular compositional context and require greater insight into their evaluation and design. However, more generally, it can be seen that three principal relationships can exist between associated goals that may result from contributing autonomous components in DSoS. First, there are complementary goal-relationships that have a reinforcing affect upon each other. As an example consider integrating two systems that target the same market base. Such relationships require little, if any, recomposing (and cannot only be readily integrated) but increase the dependability and value of the emergent service. One of the requirements of the EU EPS is a common market environment (open market) where all participating systems adhere to the laws of trading and competition. Secondly, in some situations, whilst goal-relationships may not be complementary, in a pre-existing sense, they may still be compatible – in terms of being more readily reconcilable with each other through conceiving alternative compositional approaches to achieving an emergent service. Compatible communication protocols would allow integration between components. Continuous monitoring of the performance could reveal further information regarding embedded goals. Lastly, and inevitably, there exist conflicting goal relationships. It is important that such relationships between autonomous components are identified early so that increased compositional effort can be focused on these areas to influence either the appropriate selection of alternative (and more accommodating) components or to ensure that such relationships have the major influence on the architecting of the envisaged coordination system. In some situations this may involve configuration of conflicting goals into higher or lower prioritisation status.

Definition of Emerging Service

Since we have talked about the need to compose the emerging service according to some rationale, we should stress that the rationale of such a composition should stem from the

regulations of the emerging service. We identified during the development of the TA case study that the emerging service is more thab just a front end between the consumer and the service providers (autonomous systems), that manages transactions via exception handling schemes. Similarly in the EXaMINE case study a European Wide EPS cannot be a network of interconnected national grids. There has to be regulation regarding its operation. Regardless of the time of its physical existence it needs to be regulated in terms of scope and operational policies. It is only then that intelligent composition can really take place. The scope of the emerging service determines the type of consumer. Immediately this enables the composition process to reason about the type of autonomous systems that can be utilized since the consumer types would need to be similar. Also operational policies that regulate how the emerging service should be provided would allow not only to reason about the compositional semantics but also determine the type of functionality the SoS would need to provide. Composing an emerging service it has been proved not to be as easy as putting together (or even wrapping) methods borrowed from autonomous systems. Additional consideration is needed when the SoS 'adopts' a method from a component system in order to carry out a service. Although wrapping allows us to an extent control the I/O of the method it does not resolve problems stemming from the mismatches between operational policies that may exist between the SoS and the autonomous systems. An example of this can be the cancellation method of hotel booking systems. Although a large number of them would only require the booking reference number, policies regarding cancellations varies considerably. All sorts of problems arise when these policies do not match the policies of the SoS namely maintaining the state, coordinating the transactions (as we shall see later), carrying out compensation etc.
Let us now take the extreme view (as in EXaMINE) that the service or product of each autonomous system is of strategic importance. The EXaMINE case study has been looking into composing national grids into a network of networks or a SoS that manages power distribution. The regulation process would have to be managed by all participants since it directly affects national interests. Whereas the TA could remain centrally controlled in the case of EXaMINE control and balance of power are the foremost causes of "failure". Whereas in the TA organizational failure manifested itself as data inadequacies, incompatibilities etc, in the EXaMINE study these can result in political

disputes. We therefore argue that the emerging service itself needs to be regulated. In technical terms, policies regarding scope and operations would need to be represented in data structures that would in turn be used to reason (accept or reject) certain configurations. This need is even more apparent when one considers evolutionary aspects of autonomous systems. We need some way to capturing those and making sure that the composition of the emerging service from individual autonomous systems (and therefore their policies) reflects what the emerging service is regulated to provide.

Operations / Transactional Handling

The travel agent case study [DMS3] is concerned with multi-party transactions which are distributed over many locations and which may require a considerable time to complete. Each party in such a transaction has a set of pre-conditions and a set of post-conditions which must be met before the transaction is judged to have been successful from that party's point of view. Thus, for a transaction to be judged to be well formed, the evidence, embodied in a set of instruments, must reliably reflect the intended acts of remote parties. For this to be the case, there are three characteristics of the instruments and the operations on them which must be assumed:

Atomicity: specific actions occur exactly once or not at all and the parties are able to confirm completion of an action.
Persistence: once information is generated it does not disappear; it may be changed, but the instrument(s) must record the original and the changed information.
Security: which, in this case, also refers to the authenticity and integrity of information represented in instruments.

The TA case study is structured using CA actions[Zorzo *et al.* 2003]. We use multiparty transactions to carry out specific functions such as 'booking', and we place these within a context (which can be thought of as a box) inside which exception handling occurs. The problem that was raised with this type of structuring is handling long term transactions. Although in the majority of cases the 30 second time-out rule employed by some architectures is adequate to deal with network delays, there are cases where the nature of

the transaction is such that it requires a much longer time span. The problems that arise by in situation are several.

Consider the following example. We have mentioned that the SoS TA allows us to book a full trip by conducting the appropriate methods of booking systems. The autonomous booking systems operate their individual cancellation policies. By placing all transactions that handle availability checking, booking, payment and cancellation within one context (as a multiparty transaction) we are facing a serious problem; that of the state. Cancellation concerning a trip or an item of a trip, is an action that has a large time span. Hotels may allow 2 days after initial booking has been made. This poses the question of how do we handle such a transaction while at the same time maintain it as part of the same context; that of buying a full trip.

By placing the cancellation action outside the context we immediately need to make arrangements for maintaining certain data about the user, the trip and booking systems we used etc. by maintaining the action inside we need to maintain the box i.e. its state. The state however is not controlled by the SoS since booking systems work independently.

We need therefore to recognize that with transactions that cross multiple domains of management notions such as persistence and atomicity cannot be assumed.

There are two configurations of the relationships of a multi-party transaction at the structural[1] level:

- A centralised transaction monitor in which each of the participants has a direct relationship with one particular participant in the role of transaction manager. The logical point of co-ordination is also a physical point of control.
- Distributed transaction management, in which each participant undertakes transaction management responsibilities and the logical point of co-ordination is, in fact,

[1] In this discussion, the term "structural" applies to the roles and responsibilities of the actors directly participating in the DSoS while "infrastructural" roles and relationships are those associated with the deployment of reusable, generic resources which are exploited in the execution of structural relationships.

replicated and distributed.

In the first approach, which is implemented by transaction monitoring functionality, all transacting parties must have a pre-defined relationship with one particular party responsible for the co-ordination. “Pre-defined” here means that these relationships were established outside the context and infrastructure in which the transactions will be executed. In the second, which is implemented in distributed transaction management, each party depends on all the others and must be able to monitor and interpret their acts. These two approaches to the allocation of responsibilities in a distributed transaction result in a different relationship between structural and infrastructural responsibilities. Atomicity of operations, persistence of information and security, authenticity and integrity of messages are dependabilities or qualities of service which are delivered at the structural level either as end-to-end or centralised mechanisms.

In the case of a distributed transaction, the economies of provision are quite different. Since each participant takes responsibility for components of the transaction and needs to be able to monitor remote activities and states, each needs to be able to rely on the quality of a set of service and applications components within their own domain and in each of those of the other participants. In this case, the pre-established relationship must be with the infrastructural suppliers and it is possible that the transacting parties are establishing a new context as well as a new instance of commerce. In this case, new instruments, which arise from the characteristics of the new context, may well be required.

In the distributed approach to transaction management, and here we are concerned not merely with distribution over time and space but, more significantly, distribution over the boundaries of different enterprises, each enterprise must have the option and capability of replicating all of those aspects of transaction co-ordination which are relevant to their particular interests. They must also be able to rely on the provider of the infrastructural environment to ensure that their view of the current state of any transaction is coherent with the views of all the other participants of that transaction. Thus, atomicity, persistence and security become responsibilities of the environment provider and

infrastructural in nature, and it is these qualities of service and application which dictate the characteristics of the instruments of the structural conversations.

Evolution

A SoS is built by integrating systems which might be under the control of organisations totally separate from that commissioning the overall SoS. (We will refer to the existing (separate) systems as "components" although this must not confuse the question of their separate ownership). In this situation, it is unrealistic to assume that all changes to the interfaces of such components will be notified. In fact, in many interesting cases, the organisation responsible for the components may not be aware of (all of) the systems using its component. One of the most challenging problems faced by researchers and developers constructing *dependable* systems of systems (DSoSs) is, therefore, dealing with on-line (or unanticipated) upgrades of component systems in a way which does not interrupt the availability of the overall SoS. It is useful to contrast evolutionary (unanticipated) upgrades with the case where changes are programmed (anticipated). In the spirit of other work on dependable systems, the approach taken here is to catch as many changes as possible with exception handling mechanisms. Dependable systems of systems are made up of loosely coupled, autonomous component systems whose owners may not be aware of the fact that their system is involved in a bigger system. The components can change without giving any warning (in some application areas, e.g. web services, this is a normal situation). The drivers for on-line software upgrading are well known: correcting bugs, improving (non-) functionality (e.g. improving performance, replacing an algorithm with a faster one), adding new features, and reacting to changes in the environment. When a component is upgraded without correct reconfiguration or upgrading of the enclosing system, problems similar to ones caused by faults occur, for example: loss of money, TA service failures, deterioration of the quality of TA service, misuse of component systems. Changes to components can occur at both the structural and semantic level. For example changes of a component system can result in a revision of the units in which parameters are measured (e.g. from Francs to Euro), in the number of parameters expected by an operation (e.g. when an airline introduces a new type of service), in the sequence of information to be exchanged between the TA and a

component system (e.g. after upgrading a hotel booking server requires that a credit card number is introduced before the booking starts). In the extreme, components might cease to exist and new components must be accommodated.

Although there are several existing partial approaches to these problems, they are not generally applicable in our context. For example, some solutions deal only with programmed change where all possible ways of upgrading are hard-wired into the design and information about upgrading is always passed between components. This does not work in our context in which we deal with pre-existing component systems but still want to be able to deal with interface upgrading in a safe and reasonable fashion. Other approaches that attempt to deal with unanticipated or evolutionary change in a way that makes dynamic reconfiguration transparent to the TA integrators may be found in the AI field. We believe that we need to use fault tolerance as the paradigm for dealing with interface changes: specific changes are clearly abnormal situations (even if the developers accept their occurrence is inevitable), and we view them as errors of the integrated DSoS in the terminology accepted in the dependability community. Error detection aims at earlier detection of interface changes to assist in protecting the whole system from the failures which they can cause. For example, it is possible that, because of an undetected change in the interface, an input parameter is misinterpreted (a year is interpreted as a number of days the client is intending to stay in a hotel) causing serious harm. Error recovery follows error detection and can consist of a number of levels: in the best case dynamically reconfiguring the component/system and in the worst with a safe failure notification and off-line recovery.

We also believe that we need a structured approach to dealing with interface changes that relies on multilevel exception handling which in turn should be incorporated into a DSoS. Promoting multilevel structuring of complex applications to make it easier for developers to deal with a number of problems, but our main focus here is structured handling of interface changes. The general idea is straightforward [Cristian 1995]: during DSoS design or integration, the developer identifies errors that can be detected at each level and

develops handlers for them; if handling is not possible at this level, an exception is propagated to the higher level and responsibility for recovery is passed to this level. In addition to this general scheme, study of some examples suggests classifications of changes which can be used as check lists. At this stage we would also like to acknowledge the need for communicating semantic information between component systems. Being able to communicate additional semantic information may resolve some of the conflicts presented in PCE4 but also enable better handling of interface upgrades. In the initial stages we found that in order to communicate semantic information between two components or in the case of the TA between the SoS and its providers we need a structured collection of information (meta-data) as well as a set of inference rules that can be used to conduct automated reasoning. Traditionally knowledge engineering [Hruska & Hashimoto 2000], as this process is often called, requires all participants to share the same definitions of concepts. In our case, for example, definitions of what is a trip or a flight as well as the parameters for each of these have to be defined and shared. The protocol for booking and paying for a trip or an item is also required. Detailed descriptions of the parameter types and their semantic information also need to be held in a shared knowledge base. Knowledge bases however and their usage does not necessarily make the system more flexible; quite the contrary. Requests would have to be performed under strict rules for inference and deduction. The SoS would have to process its metadata (globally shared data descriptions) in order to infer how to make a request for a particular method (i.e. booking a flight) and further more infer what parameters accompany this method and what is their meaning.

4. Conversation Theory a Framework for analyzing SoS's

Having discussed our results and conclusions drawn from the two case studies it is apparent that what distinguishes a SoS from a distributed component system is the fact that *control* is also distributed. When control is distributed and in particular when there is no central authority to govern and take decisions then conflicts can arise and unexpected changes may take place. The question we ask in the rest of the document is whether some of these issues can be caught early in the pre-requirements engineering process. In order

to do something like this we would need to carefully explore this new problem space using new concepts that abstract from typical UML driven solution and design patterns. The theory of conversations [PP2000] is posited as (among other things) a tool to analyse social, commercial and organizational roles and relationships. The term 'conversation' is used in the normative sense to describe a relationship between two roles. Each conversation is described by attributes such as: significance, mutuality, capability and control. We do not attempt at this stage to provide transactional properties. We do however attempt to understand the nature of transactions at this abstract level. The aim of course is to detach the analysis from any bias on certain technologies and produce a list of requirements regarding the relationships between all participants of the SoS based on their roles, responsibilities and conversations. Viewed at the organizational level, conversations correspond to pre-defined contractual arrangements between systems. The degree of co-operation, trust and significance of such relationships is determined by the contractual arrangements agreed between the two systems. What we are proposing here is that since we can use conversation theory to determine or describe rather the nature of these relations, we could take a step further and make use of these descriptions to derive and understand the nature of such relations. Since transactions in a sense simulate a contractual agreement we want to use conversation theory in order to make this simulation process more accurate. Conversation patterns can indeed determine transactional patterns.

We use the variable *significance* to indicate how the benefits of the relationship are being distributed. In the 'Theory of Conversations' significance can either be symmetrical (equal benefits) or asymmetrical (unequal benefits). We also use the variable *mutuality* to indicate how responsibilities are being distributed. Mutuality shows how responsible one system is for the benefits of the other. *Capability* is the variable that shows if resources are shared by two systems in order to fulfill their responsibilities. Finally we use the variable *control* to indicate which agent in a network has the power to initiate or terminate a relationship. If for example a conversation is of asymmetrical significance and mutuality is zero (i.e. none of the two component systems who are talking to each other are responsible for the other's benefit) then we could not successfully implement an

atomic transaction. Atomicity requires a level of support that operates underneath the transaction and is agreed upon by both systems who are in conversation. Zero mutuality would indicate that such support layer does not exist. In the case of the travel agent the case study was build with conversations of zero mutuality and asymmetrical significance. The same pattern arises in the second case study since national interests and national security prevail. This corresponds to the view that participating systems are autonomous and they may not even know that they are in collaboration under the umbrella we call SoS. By combining various degrees of the variables of mutuality and significance certain patterns arise. The next paragraph describes them in more detail.

Significance & Mutuality

The following table shows all the possible combinations of the four variables and the resulting relationship types. In the first instance we will consider the values of significance and mutuality. As we mentioned earlier significance can range from asymmetrical to symmetrical and mutuality from high to low. The values of both of these variables are fuzzy and cannot be successfully determined unless they are based on real evidence. For the sake of the example and the case study we assume that the variables take only absolute values. Given this, 4 patterns of relations emerge.

Conversation	Significance	Mutuality	Properties
A	(symmetric)	(high)	ACID
B	(symmetric)	(low)	Exception Handling
C	(asymmetric)	(high)	Atomic
D	(asymmetric)	(low)	Multi-layered Handling

Table 1. Travel Agency Example

In terms of significance and mutuality we distinguish between symmetric (benefits are equally distributed and both parties a equally responsible) and asymmetric relationships (one party benefits mostly and none of them is responsible for the benefit of the other).

The above table resulted in four possible combinations of significance and mutuality. Let's take each one in turn. Conversation type A is defined in terms of symmetric significance and symmetric mutuality. Both parties enjoy equal benefits from the relationship and both are equally responsible for the benefits of the other. Many times the contractual arrangements between the two systems are such that one party is exclusively used by the other. The significance of this relationship is symmetrical as benefits are equally distributed. In terms of contractual arrangements the relationship is also mutual. This is because the party who initiated the relationship cannot dissolve it, (due to contractual arrangements) while the other party is not allowed to supply to other clients/agents.

Conversation type B describes again relationships where although the benefits are equally distributed none of the parties is responsible for the benefit of the other. Again this type of relationship can be observed in a distributed environment as the one described earlier. The difference would be in the contract between the two parties where one party selects the best possible provider while the second is ready to provide its service to the best bidder. Conversation type A describes, in a sense co-operation whereas type B describes competition.

Conversation type C implies a situation where only one of the parties enjoys the benefits of the relationship although mutuality is equal. Conversation type D reveals a parent child relationship of asymmetric significance and mutuality. It implies a situation where one party is putting the effort for the benefit of the other. In the travel agent case study all relationships between the SoS and its sub systems are of symmetric significance since both parties enjoy the benefits of such relation although they are not mutually responsible. In symmetrically significant relationships although responsibility may not be present (i.e. low mutuality) both parties would be interested in maintaining the state of affairs as long as the relationship is beneficial. In the case of the EXaMINE case study this would directly related to the democratic accountability of the regulatory body (i.e. EU) and how democratically "power" and "authority" is distributed within the SoS.

The variable *control* indicates which of the parties in a conversation can initiate a conversation. In a distributed SoS control indicates which of the parties have higher authority or have a better position in a competitive market. In the case of the travel agent it is the SoS that has higher control since it is capable of initiating and terminating a transaction with its sub-systems. While in the case of EXaMINE control would have to remain equally distributed.

Capability relates to the system's ability to access resources. *Low* capability indicates that both components are responsible for acquiring their own resources. When capability is *high* then componets may have agreed upon using or sharing resources. In the travel agent example again capability is always *low*. This reflects the fact that the SoS although it has authority over initiating transactions it does not have access to the sub-systems databases. And this indicated the strong boundary that exists between the travel agent SoS and the participating booking systems. On the hand capability in the EXaMINE project is *high* since there is (at least this is the requirement) to access components of the system that may cross national boundaries. For example there is a requirement that all statistical data from the various national grids are available across the network. Since however control is presumed to be equally distributed decisions regarding initiating and terminating such a conversation is kept at the component level (national level)

Since we can classify the relationships that appear within the context of the TA and EXaMINE case studies using conversations we could also use conversation descriptions to derive transactional properties that can be supported.

5. Role and Responsibility Driven Analysis

Putting systems together for the purpose of developing new systems that offer additional services requires some additional analysis. The 'owners' of these new services need to share the responsibilities of the new services which may indeed be collective although the degrees of responsibility change and at the same time it may not always be clear. Each of the systems will carry on performing a number of tasks as it did in each autonomous state (the state may or may not change). We call the tasks performed as part of the SoS duties

and so each system that is part of the system of systems has to perform its duties in order to maintain the 'new' services running. By studying the TA case study we can identify three combinations of duty-responsibility patterns that have different implications with regards to implementation. Outside the scope of the TA we can identify 4 patterns in total. As the examples later will illustrate it can be the case where duties are shared between two systems although responsibility of those duties lies within one single system. It can also be the case where duties are shared and responsibility is collective. Another case arises when responsibility of service and the duties that support this service remain within the boundaries of one system. Finally the are cases where the responsibility and duties may reside in completely separate different systems. Bear in mind that the '"emerging service" is the outcome of putting autonomous systems together. It cannot exist without the collaboration of them all. In previous paragraphs we talked about the need for exploring this new problem space using concepts such as roles, responsibilities and conversations. By doing this we are building enterprise models. Before we delve into the analysis of these concepts here are some definitions.

Definitions

Enterprise models represent the concept of division and allocation of responsibilities to create a network of roles interacting through conversations. A Role can be considered as a composition of responsibilities which defines the boundaries of a system. A conversation between roles represents the context of a single or list of transactions for the generation and interpretation of information. Resources represent domains of ownership or instruments of communication and exchange.

Analysis-Roles

Roles are, as the definitions suggest holders of responsibilities. The first step of our analysis requires breaking down the participants into roles capable of holding some responsibility. Responsibilities are broken down into duties which represent the tasks that roles need to perform in order to maintain a state of affairs. This would be agreed by the SoS regulator. The purpose of breaking down autonomous systems into roles is to examine whether certain roles are capable of holding certain responsibilities, the tasks

involved and boundaries that may exist between responsibility holder and duty execution (which is a common pattern in TA, more later). In the case of the EXaMINE case study the boundaries are political, social and economic as we showed in PCE4. In other system i.e. merging banks or large insurance companies the boundaries may primarily be legal. In the case of the TA the major boundary is organisational since it hinders information exchange and capability to share and access resources. The question that should be raised in this initial stage is whether roles have enough resources to execute their duties, what are the dependencies amongst the execution of the tasks as well as what sort of compensation schemes the SoS needs to support in order to handle failures.

Analysis-Responsibilities

The responsibility analysis is important in clarifying some of the implementation issues of the system. In this process we identify new services as well as the tasks for carrying out those services. We call these tasks *duties* since they are executed by roles who are the primary responsibility holders. Every duty is itself a process that may or may not carry a number of dependencies. When for example system *a* sends a request to systems *b* we can assume that system *b* has to be in a state *ready* to accept this request and consequently respond to it. There are a number of dependencies like this that can alter the way a service is executed and the way the system of systems is implemented. Each responsibility as we mentioned is associated with a number of duties, tasks. Given the nature of the task we can select an appropriate role to hold responsibility for these tasks. For example the TA is responsible for accepting a request from the User. So the TA should provide a form for accepting the User's requirement. The TA is also responsible for sending this requirement to the reservation system. The reply to the user however, is collectively a responsibility of both the reservation system and the TA. The reservation system for example needs to be on line when the TA sends the request. Additionally it has to respond within the timescale posed by the TA. The reservation systems (since it is out of the control of the TA) can move from an online state to an offline state independently. The technical implication this will have is that the TA prior to sending a request to a specific reservation system has to check its state. It is fair to conclude at this point that the whole notion of responsibility analysis is derived by the lack of control the

SoS has on the behaviour of its individual components. Since there is a lack of overall control in the SoS, responsibility of service cannot be assumed exclusively by an individual component. This is something that will impact and should be considered when the communications infrastructure is being developed. From the analysis of the TA we have concluded that the impact of this lack of control will result in additional transactions between the various components in order to assess the state of each component at all times during the provision of the new service. The following diagrams show how responsibilities are shared for carrying out three basic functions. Notice the patterns we mentioned earlier regarding responsibility and duties. Some of them are shared between two systems while others remain central. The new service itself is being offered by a combination of all systems, although responsibility and duties change hands as the SoS moves in different states.

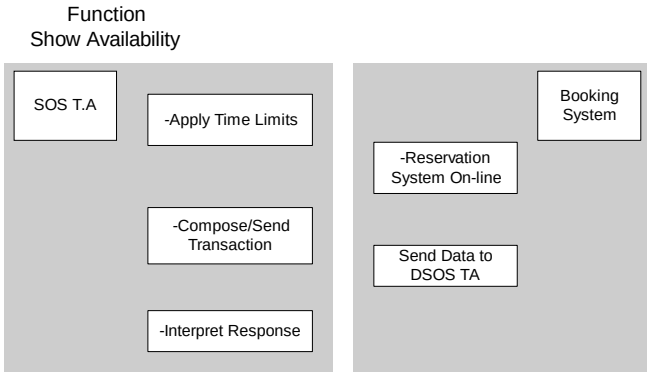

Figure 1. Shared Duty-Single Responsibility

The above diagram shows a basic function, the implementation of which has been broken down to 5 duties. The TA is applying a time limit within which it expects the request to be answered. It checks that the reservation system is on line (something it has no control of) and it composes and sends the transactions. The reservation system (assuming it is on line) sends a response which is then interpreted by the TA and presented to the user. Although transparent to the user the operation of this task is dependent on the reservation system. The gap between the two grey panels denotes organizational boundary. The responsibility however for performing such a task is passed to the travel agent. In

implementation terms the TA would be expected to direct the request to a different reservation system if one does not respond within the time limits or it is not available. Consider the following diagram.

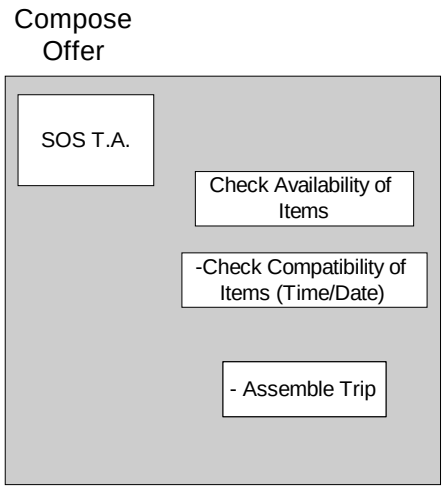

Figure 2. Single Duty-Single Responsibility

The above function involves the composition of the items collected by all systems into full trips. The responsibility for the composition and its sub-tasks is passed to the TA. At this stage of booking, the reservation systems are not involved at all so the task of offering full trips to the users is implemented entirely by the TA. There are no dependencies at this stage between the TA and its sub-systems. Consider now the following diagram.

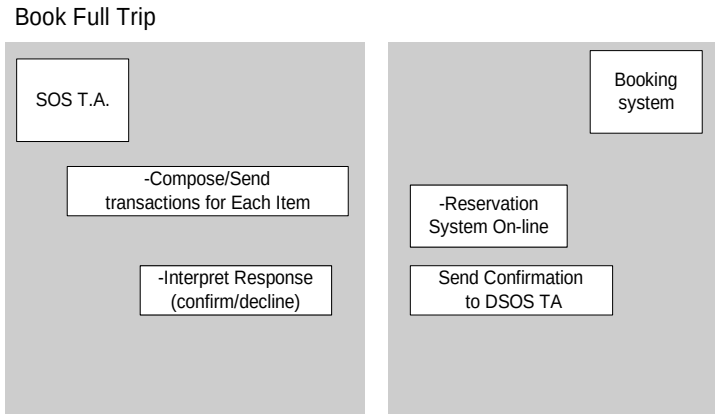

Figure 3. Shared Responsibility-Shared Duty

The function of booking the trip is entirely dependent on the reservation systems that offered the items in the first place and the responsibility of booking is also shared. The tasks or duties involved are composing and sending the transactions to each of the systems involved (flight, accommodation, car). Assuming that the systems are online then they should respond to the booking request and finally the TA should interpret the response and inform the user. It was pointed out earlier that the responsibilities of the individual systems involved accepting a request for availability or request for booking and responding to them either by showing availability of confirming a booking. The TA is the medium between the user and the reservation system that actually owns the items on offer. Although it is the responsibility of the TA to compose and send the transactions to its sub-systems it is not responsible for making the actual bookings. Although the TA may not have overall control, it can monitor the state of its sub-system and take appropriate action. The example above however may result in a situation that the TA cannot monitor. Consider that a booking of a full trip that consists of flight, accommodation and car rental would require the TA to transact with 3 systems, and get confirmation for all 3 items in order to confirm the booking to the user. Some of the situations that may arise are:

- One of the reservation systems confirms a booking without actually making a booking due to a fault.
- One of the reservation systems becomes unavailable
- One of the reservation systems makes item unavailable
- The TA gets confirmation for 2 out of the 3 items of the trip since the 3rd became unavailable.

All of these situations can result in only part of the trip being booked because one of the items has either being booked by some other system or the system is now offline. The user's requirement of a full trip is incomplete and therefore cannot be satisfied. A booking of two items of a full trip has been completed and are now unwanted. The user

cannot be charged since the trip is incomplete. The reservation system that went offline or made the item unavailable cannot be charged since it is not aware of its existence as part of SoS TA. Two of the reservation systems responded to the request for booking and confirmed the booking of the items requested. These systems are expecting payment.

Analysis-Allocating Responsibilities

We have showed how *conversations* are used to characterise relationship between the two systems. We also discussed the importance of the variables of control, capability significance and mutuality. Here we suggest that the system which has *control* of initiating a conversation is also responsible for that conversation. The TA case study accepts that the reservation systems involved are independent systems that do not communication with each other. The TA has to contact them in turn to build the trips offered to the user. Consider this diagram that shows shared duty/shared responsibility figure.

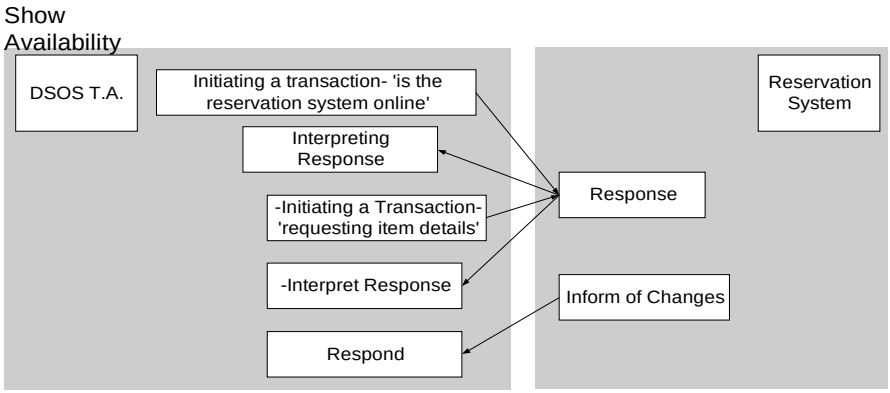

Figure 4. Sharing Responsibilities

All the transaction types are featured in this diagram. The SoS TA is initiating a transaction to identify whether the reservation system is online. The reservation system responds to the request. The SoS TA interprets the request and initiates a new transaction, requesting details of items on offer. The reservation system again responds and the SoS TA interprets that response. One could argue here that the response of the reservation system is responsibility of that system. The reason we distinguished responses and

initiations is to identify where the control of the conversation lies. When the TA initiates a transaction it would be responsible for interpreting the response. The response however could be ‘no message’. The reservation system may not reply at all to the initiation so the TA has to interpret this situation and take action. If the reservation system does not respond to an initiation then the TA who maintains control of the dialogue could request the services of some other system.

Responsibility is held by the system is initiating, or interpreting a transaction. In the diagram above we have a dialogue between two systems. The SoS TA is attempting to identify what are the items on offer of the reservation system. It is initiating the conversation and but is also responsible for interpreting the responses. The interpretation duty may have to deal with no response at all.

The reservation system at some point is initiating a transaction to inform of changes to the policies regarding the items on offer. It is therefore its responsibility of initiating this conversation and therefore informing the SoS TA of changes. So we share responsibilities according to which system owns the attribute of control. The attribute of control is not static but it can change dynamically as the example illustrated. It is therefore essential for both systems to be able to interpret these transactions initiated by both sides. In the case of the travel agent case study the SoS TA would need to be able to accept a transaction that informs of changes in policies or other details.

The above figure illustrates some of the technical implications of this analysis. It suggests that the TA will be able to send requests to the reservation system via a specified interface, interpret its responses and furthermore deal with its exceptions. The issue of interpreting transactions initiated by the reservation system refers to the problem of dealing with exceptions thrown by the sub systems to the level above. Building interfaces to communicate with a system and maybe a wrapper to restrict its output it is not sufficient to implements a system such as the TA.

Developing a service that is transparent to the user does not imply only building transactions according to the interface specifications but also dealing with exceptions. If the application was to be entirely transparent the TA should be able to interpret transactions initiated by the reservation systems before presenting them to the user. This of course requires additional effort in analyzing the participating systems in depth and not just at the interface level in order to identify all possible exceptions. This process would help in building the appropriate exception handling mechanisms at the TA level. One would have to deal with the exceptions thrown by the sub system and decide whether to interpret them or pass them directly to the system above which is our case is the user.

6. Conclusions

In this deliverable we presented a number of issues that were raised primarily from the TA case study and to a lesser extend by EXaMINE. Although there is a lot of literature on architecture and design we felt that there is little of work available to address SoS specific issues. The most important of those relate to the way the emerging service is composed, regulated, delivered (transactional model) and how it evolves. Since there are a number of tools and methodologies addressing structuring techniques and design patterns we concentrated on concepts that would allow us to look at this new problem space that a SoS addresses.

We showed how boundaries that surround various systems can have an immediate affect in the way the system is perceived, developed and consequently delivered. One of the most important attributes we showed to be that of control. This is what distinguishes DSoS from other component based distributed system. By control we mean authority to regulate and impose structure and operational policy of the emerging service, but also to foresee future behavior of a system, initiate and terminate transactions as well as access to all component of a SoS.

Given this additional dimension we believe that a responsibility driven analysis where systems are viewed as roles holding responsibilities would enable developers or

regulators of a SoS to capture certain type of non technical failures (organizational failures) early prior to the requirements analysis stage.

7. References

[AL2001]Axel van Lamsweerde, Goal oriented requirements engineering: A guided tour. In Proceedings, RE'01 5th IEEE international symposium on Requirements Engineering pp 244-263,Toronto August 2001

[CORBA] Jon Siegel, Corba 3:fundamentals and Programming 2nd edition, OMG press, John Whilley and Sons, 2000

[DMS3] P. Periorellis, J.E. Dobson. Case Study Problem Analysis. The Travel Agency Problem. Technical Deliverable. Dependable Systems of Systems Project (IST-1999-11585). University of Newcastle upon Tyne. UK. 37 p. 2001. www.newcastle.research.ec.org/dsos/

[EXAMINE1] Ferrante A, Diu A 2002. Needs Expression :Revised Version. Technical Deliverable. EXaMINE Project (IST-2000-26116).

[HH2002]Hruska & Hashimoto 2000 T. Hruska and H. Hashimoto (eds), Knowledge Based Software Engineering, Ios Press June 2000.

[JAVA RMI] Java™ Remote Method Invocation (RMI), technical documentation, Sun Corporation, http://java.sun.com/j2se/1.4/docs/guide/rmi/, last accessed October 2003.

[MD1999] Martin M & Dobson J E Theory of Conversations, Center for software Reliability, 1999 Technical report

[PCE4] Dobson J, Periorellis P, 2002 Models of Organisational Failure. Technical Deliverable. Dependable Systems of Systems Project (IST-1999-11585). University of Newcastle upon Tyne. UK. www.newcastle.research.ec.org/dsos/

[PP1999] Periorellis P, Dynamic Enterprise Modelling, Ph.D. Thesis, Center for software Reliability, 1999

[PD1999] Periorellis P, Dobson J E Enterprise Modelling in Context, Conference proceedings CIMCA99' February 14-16 1999 Vienna Austria.

[PJD2002] Periorellis Panayiotis, John E. Dobson *On Enterprise Modelling Dynamics, Towards E.M. Formalisation*, proceedings 9th European Concurrent Engineering Conference, pp 111-115, April 15-17, 2002 Modena Italy

[PD2002] Periorellis Panayiotis, John E. Dobson, *Organisational Failures in Dependable Collaborative Enterprise Systems*, in Journal of Object Technology, vol. 1, no. 3, special issue: TOOLS USA 2002 proceedings, pages 107-117. http://www.jot.fm/issues/issue_2002_08/article7

[PP2002] Periorellis P, *Enterprise Modelling Dynamics*, proceedings of 6th World Multi-conference on Systemics, Cybernetics and Informatics, July 14-18 2002, Orlando Florida

[RPZ2003] A. Romanovsky, P. Periorellis, A.F. Zorzo, On Structuring Integrated Web Applications for Fault Tolerance. To be presented at ISADS'2003 Pisa, Italy April 2003 http://www.cs.ncl.ac.uk/research/trs/index.html

[Zorzo *et al*. 2003] A.F. Zorzo, P. Periorellis, A. Romanovsky. Using Coordinated Atomic Actions for Building Complex Web Applications: a Learning Experience. Presented at *the 8th IEEE International Workshop on Object-oriented Real-time Dependable Systems* (WORDS 2003), January 15-17, 2002, Guadalajara, Mexico. To be published by IEEE CS. 2003.