Managing evolution risk of cross-organizational services

VON SUSANI, Odorico

Abstract

The world's resource whose production, transport, refinement and consumption have increased incredibly more than all other ones is information. Organizations produce, exchange and consume more information every second. They need to share data with an increasing number of partners to manage their operations. This means that an organization must often adapt its information systems such that it can work with a new partner or because its current partners require it. These necessary adaptations must be made while the organization continues to operate and without causing any damage or disruptions. Thus, each organization must allocate an amount of resources to adapt its applications when required by its changing external environment. However, it may not be easy for an organization to know how many resources it needs for this. This is because of the multiplicity of dependencies and because of the uncoordinated nature of changes among all organizations. In this thesis, we propose a model to determine how many resources an organization needs to adapt to its evolving world. We also show how an organization can price an extension of [...]
Managing Evolution Risk of Cross-Organizational Services

THÈSE

présentée à la Faculté des sciences de l'Université de Genève
pour obtenir le grade de Docteur ès sciences, mention informatique

par

Odorico VON SUSANI
d'Autriche

Thèse N° 4375
Doctorat ès sciences  
Mention informatique  

Thèse de Monsieur Odorico VON SUSANI  

intitulée :  

"Managing Evolution Risk of Cross-Organizational Services"  

La Faculté des sciences, sur le préavis de Messieurs D. BUCHS, professeur associé et co-directeur de thèse (Département d’informatique), Ph. DUGERDIL, professeur et co-directeur de thèse (University of Applied Sciences – Geneva School of Business Administration - Département of information systems – Geneva, Suisse), L. FERREIRA PIRES, professeur (University of Twente – Faculty of Electrical Engineering, Mathematics & Computer Sciences – Enschede, The Netherlands), et B. HIRSBRUNNER, professeur (Université de Fribourg – Département d’informatique – Fribourg, Suisse), autorise l’impression de la présente thèse, sans exprimer d’opinion sur les propositions qui y sont énoncées.

Genève, le 18 novembre 2011

Thèse - 4375 -

Le Doyen, Jean-Marc TRISCONE

N.B.- La thèse doit porter la déclaration précédente et remplir les conditions énumérées dans les "informations relatives aux thèses de doctorat à l’Université de Genève".
Aknowledgments

I would like first to thank Prof. Philippe Dugerdil for the opportunity he gave me to do this thesis under his supervision. I would also like to thank him for all the support and counseling that he provided me all these years and without which this thesis would probably not exist. I would also like to thank him for the trust and the friendship he has shown to me for all these years. The numerous discussions he had with me on this subject helped me to find my way out of this labyrinth, which was for me a difficult one.

I would then like to thank also Prof. Didier Buchs for accepting to co-supervise this thesis and also for the different reviews and follow-ups he provided me during this period of time.

I would also like to thank in advance the members of the jury, for the time they took to read, review and judge this thesis and its presentation.

I would then like to thank my family, especially my wife Tanya and my four children Milena, Odorico, Ludovic and Maia who encouraged me in my most doubtful moments.

I would then like to thank all my friends, who supported my talking about my ideas without ever showing any sign of impatience for so many years. I would especially like to thank Patrick Varone for his help on the Java version of the earliness-tardiness algorithm used in this thesis.

I would then like to thank Georges, who was always convinced of the success of this thesis and who helped me to understand myself.

I would like also to thank my past and present employers that always supported my efforts and provided me with some time such that I could advance in my research. I would also like to thank my colleagues that have always been supportive of all my efforts even though they could not understand why I was doing all this.

I would also like to thank my mother, for all the sandwiches she provided me with for this thesis to exist. Finally, I would like to thank my father, but a little less than all the others for harassing me relentlessly since 35 years to get me to do this PhD.
For my father, as he commanded me to do.
Sommaire

Notre monde s'interconnecte de plus en plus. La croissance de l'économie globalisée a été nourrie ces dernières décennies par l'augmentation des échanges de matières premières, de biens et de services entre les organisations et les entreprises de toutes les régions du globe. Mais la ressource qui a vu croître plus que toutes les autres ressources, sa production, son transport, sa transformation et sa consommation est l'information.

Les organisations produisent, échangent et consomment plus d'information à chaque seconde qui passe. Leurs systèmes d'information deviennent de plus en plus complexes au fur et à mesure qu'une part toujours plus grande de leurs activités en dépend. Dans le cas où des organisations externalisent une partie de leurs activités à d'autres organisations, elles doivent toujours continuer à se coordonner avec ces partenaires pour pouvoir continuer leur propres activités. Cette coordination est effectuée à travers l'échange d'informations entre les organisations. Les organisations offrent généralement des services qui permettent l'échange d'information avec d'autres organisations.

La compétitivité et la spécialisation tendent à faire croître le nombre de partenaires avec lesquels une organisation est amenée à échanger des données. De plus, la globalisation exacerbe cette compétition. Ceci augmente la fréquence à laquelle une organisation subit des changements dans ses activités. Par conséquence, cela augmente aussi le nombre d'adaptations qui doivent être apportés aux systèmes d'information de ces organisations et de leur partenaires pour que leurs activités respectives puissent continuer.

Ces adaptations des systèmes d'information sont nécessaires pour la bonne marche des activités d'une organisation. Elles ne doivent pas causer de dommages à l'organisation ni être la cause d'une interruption de ses activités courantes. Il est donc nécessaire qu'une organisation alloue une partie de ses ressources pour effectuer ces adaptations. Néanmoins, il est difficile pour une organisation de déterminer combien de ressources doivent être allouées à ces tâches d'adaptation. En effet, dans le cas d'un changement décidé par l'organisation elle-même, il est relativement facile pour elle d'estimer la quantité de ressources nécessaires pour effectuer les adaptations. Dans ce cas, c'est aussi l'organisation qui décide quand ces ressources doivent être disponibles.
Cependant, dans les cas où les changements sont initiés par les partenaires de l'organisation et non par l'organisation elle-même, il est très difficile de quantifier la quantité de ressources dont l'organisation aura besoin pour effectuer les adaptations. En effet, les partenaires d'une organisation ne sont pas coordonnés entre eux.

Dans cette thèse, nous proposons un modèle qui permet de déterminer la quantité de ressources nécessaires dont a besoin une organisation pour adapter ses applications lors de changements induits par ses partenaires. Au moyen de ce modèle, une organisation peut:

- Estimer la probabilité de ne pas pouvoir adapter ses applications lors de l'évolution des services de ses partenaires.
- Estimer son manque de ressources permettant d'adapter ses applications sans dysfonctionnements lors de l'évolution des services de ses partenaires.
- Estimer les coûts probables induits par ce manque de ressources.
- Déterminer si ce manque de ressources est structurel ou contextuel.

Nous illustrerons, au travers d'un exemple, comment utiliser le modèle pour identifier les actions qui peuvent réduire les coûts probables induits par l'évolution des services des partenaires d'une organisation. Nous montrerons également comment une organisation peut estimer la valeur d'une prolongation de la période d'adaptation contractuelle conclue avec ses partenaires pour chacun des services qu'elle utilise.
Abstract

The world is becoming increasingly more interconnected. The growth of the global economy has been fueled these last decades by the increase of exchanged commodities, goods and services between organizations from all around the globe. But the one resource whose production, transport, refinement and consumption have increased incredibly more than all other ones is information.

Organizations produce, exchange and consume more information every second. Their information systems are becoming more complex as more of their business processes are relying on them. Even if organizations outsource part of their business processes, they still need to coordinate with the organizations that are handling these processes for them. Thus, organizations use information services from other organizations all around the planet through the Internet or dedicated networks. They need to share data with their partners to manage their operations.

With the competition and specialization trend, the number of partners with which an organization exchanges data is rising. Furthermore, global competition induces rapid changes in an organization's business and those of its partners. This means that an organization must often adapt its information systems such that it can work with a new partner or because its current partners require it.

These necessary adaptations must be made while the organization continues to operate. They should not cause any damage or disruption to the organization's current running operations. Thus, each organization must allocate an amount of resources to adapt its applications when required by its changing external environment. However, it may not be easy for an organization to know how many resources it needs for this. This is because of the multiplicity of dependencies and because of the uncoordinated nature of changes among all organizations.
In this thesis, we propose a model to determine how many resources an organization needs to adapt to its evolving world. Using this model, it is possible to:

- Estimate the probability of disruptions induced by the evolution of used external services.
- Estimate the probable resource shortage to avoid disruptions induced by the evolution of used external services.
- Estimate the probable costs induced by the resource shortage.
- Estimate if the probable resource shortage is contextual or structural.

Through an example, we show how the model can be used to identify actions that can reduce the probable costs of future evolution shocks faced by an organization. We also show how an organization can price an extension of the contractual adaptation time of its used external services.
Table of Contents

Aknowledgments i
Sommaire iii
Abstract v

1 Introduction 1

2 Introduction to distributed computing 12
  2.1 Definitions.................................................................12
  2.2 System types..............................................................12
  2.3 Client and server.......................................................14
  2.4 Interactions...............................................................15
    2.4.1 Synchronous interactions........................................16
    2.4.2 Asynchronous interactions......................................19
  2.5 Workflows...............................................................19
  2.6 Distributed system architectures.................................20
    2.6.1 Transactional Middleware........................................21
    2.6.2 Procedural middleware..........................................22
      2.6.2.1 RPC advantages.............................................24
      2.6.2.2 RPC shortcomings...........................................24
      2.6.2.3 RPC extensions..............................................25
    2.6.3 Message-Oriented Middleware (MOM)........................25
      2.6.3.1 MOM advantages............................................26
      2.6.3.2 MOM shortcomings..........................................27
    2.6.4 Object-oriented middleware.....................................27
    2.6.5 Advantages of the Distributed Object Architecture........29
    2.6.6 Shortcomings of the Distributed Object Architecture......29
    2.6.7 Component-oriented middleware................................30
    2.6.8 Service middleware...............................................31
    2.6.9 Web Services.......................................................33
      2.6.9.1 Simple Object Access Protocol (SOAP).....................33
      2.6.9.2 Web Service Description Language (WSDL)..............34
      2.6.9.3 Universal Description, Discovery and Integration (UDDI)34
    2.6.10 Services advantages.............................................35
    2.6.11 Services shortcomings..........................................35
  2.7 Technologies comparison.............................................35
    2.7.1 Middleware Fragmentation......................................37
    2.7.2 Middleware shortcomings......................................37
  2.8 Transactions in distributed systems...............................38

3 Dynamic reconfiguration 39
  3.1 State of the art in dynamic reconfiguration......................40
7.2.1 Service provider only ................................................................. 108
7.2.2 Service client only ................................................................. 109
7.2.3 Service provider and service client ........................................ 110
7.3 Concept: Temporal availability properties of a service contract ...... 111
7.3.1 Service availability conversion specification .......................... 116
7.4 Notion: Service evolution scenarios : from one version to the next one ................................................................................................. 121
7.4.1 Unannounced service evolution ............................................. 121
7.4.2 Temporal coordination ......................................................... 121
7.4.3 Multiple simultaneous versions ............................................. 125
7.4.4 Implications of relaxing the synchronicity constraint ............... 126
7.5 Prerequisite: Information related to future evolutions .................. 128
7.5.1 Time period length of a service evolution probability distribution ........................................................................................................ 129
7.5.2 Collecting service evolution information .................................. 129
7.5.3 Notion: Aging of information .................................................. 130
7.5.3.1 Dangers of information aging ............................................ 132
7.5.4 Information updating policy .................................................. 133
7.5.4.1 Recollection based on the data age ..................................... 133
7.5.4.2 Collecting data based on the proportion of obsolete data ...... 135
7.5.4.3 Collecting data automatically on a regular basis ............... 136
7.5.5 Prerequisite: Service availability as a service ......................... 136
7.5.5.1 Service evolution information accuracy .............................. 137
7.5.5.2 Estimating the information quality ...................................... 137

8 Model ......................................................................................... 140
8.1 Model data ............................................................................. 140
8.2 Set-up, assumptions and hypotheses ........................................... 141
8.2.1 Adequate temporal condition .................................................. 141
8.2.2 Termination warning and new service version ......................... 141
8.2.3 Trained resources and immediate availability .......................... 141
8.2.4 Service evolution randomness and independence of evolution events ................................................................................................. 142
8.2.5 Time to adapt – worst case assumption .................................... 142
8.3 Blueprint .................................................................................. 142
8.4 Basic model ............................................................................. 143
8.4.1 Task order ............................................................................. 143
8.5 Indicators .................................................................................. 144
8.5.1 PERC-SUCCESS ................................................................. 145
8.5.2 PROB .................................................................................. 147
8.5.2.1 Computation ................................................................. 148
8.5.2.2 Calculation optimizations .................................................. 149
8.5.3 PROB-SUCCESS ................................................................. 150
8.5.3.1 Computation ................................................................. 150
8.5.4 EVOL-EXP ......................................................................... 151
8.6 Adding work stack .................................................................... 152
8.7 At present, now in the future
8.7.1 Problems to solve
8.7.1.1 Exponential number of computations
8.7.1.2 Estimation of the work stack
8.7.2 Calculating the indicators for all possible futures
8.7.3 Calculating the indicators for the most probable futures of each time slice
8.7.3.1 Total number of computations over the foreseeable future
8.7.4 Estimating the work stack
8.7.4.1 Estimation of the work stack for each probable past of a future Ft conditioned by a combination Ctw,t
8.7.4.2 Probable resource shortage indicator
8.8 Adding costs
8.8.1 Cost-based task order
8.9 Job-shop scheduling
8.9.1 Calculating costs
8.9.2 Recomputing all indicators using the cost-based ordering method
8.9.3 Probable costs
8.9.4 Probable costs indicator
8.10 Model implementation and processing environment

9 Simulated case
9.1 Initial situation
9.1.1 Organization setup
9.1.1.1 Exogeneous services
9.1.1.2 Internal applications
9.1.1.3 Provided services to other organizations
9.1.1.4 Organization dependencies
9.1.1.5 Service evolution probability distributions
9.2 Assessment
9.3 Increasing the allocated resources
9.4 Renegotiating the contract
9.5 Refining the contractual adjustment
9.6 Findings
9.6.1 Resource shortage and costs
9.6.2 Structural versus contextual shortage
9.6.3 Reduction of key-man dependencies

10 Limitations and weaknesses
10.1 Service evolution probability distributions
10.2 Adaptation time
10.3 Dependency identification
10.4 Unavailability costs
10.5 Computation time
11 Contribution, discussion and conclusion ................................. 226
  11.1 Contribution................................................................................. 226
  11.1.1 Overview of dynamic reconfiguration of distributed systems,
          cross-organizational interactions, services and service contracts........... 226
  11.1.2 Presentation of the inadequacy of distributed system technologies
          applied to cross-organizational interactions........................................... 226
  11.1.3 Definition of the service evolution challenge............................... 226
  11.1.4 Proposal of a model..................................................................... 227
  11.2 Discussion.................................................................................... 227
  11.3 Conclusion.................................................................................... 228

Appendices ......................................................................................... 229
  Appendix A: SOAP Example............................................................... 229
  Appendix B: RM-ODP......................................................................... 230

Bibliography ....................................................................................... 231
Illustration Index

Figure 1. Dependencies between organizations .................................................. 2
Figure 2. Dependencies of a single organization ................................................ 3
Figure 3. Evolving or dying services ................................................................... 4
Figure 4. Exceeded workload capacity ............................................................. 5
Figure 5. Chained disruptions ........................................................................... 6
Figure 6. Distributed system ............................................................................. 7
Figure 7. Multiple service versions simultaneously ......................................... 8
Figure 8. Termination delay ............................................................................. 8
Figure 9. Application dependency graph ............................................................ 9
Figure 10. Service termination event probability distribution ......................... 9
Figure 11. System types ................................................................................... 13
Figure 12. 2-tier and multi-tier client-server .................................................... 15
Figure 13. Synchronous interactions ............................................................... 17
Figure 14. Synchronous uncompleted interaction ............................................ 18
Figure 15. Asynchronous interactions .............................................................. 19
Figure 16. Middleware layer ............................................................................ 20
Figure 17. Embedded and stand-alone middleware ........................................... 21
Figure 18. Remote Procedure Call model .......................................................... 23
Figure 19. Remote Procedure Call request sequence ....................................... 23
Figure 20. MOM topologies ........................................................................... 25
Figure 21. Message sending ........................................................................... 26
Figure 22. Distributed Object Architecture ..................................................... 27
Figure 23. Method call through distributed objects .......................................... 28
Figure 24. Object Request Broker (ORB) ......................................................... 28
Figure 25. Service oriented middleware ........................................................... 31
Figure 26. Hierarchical relations between objects, components and services ...... 32
Figure 27. SOAP message structure ............................................................... 34
Figure 28. Comparison of distributed technologies [8] .................................... 36
Figure 29. Directed connection ...................................................................... 41
Figure 30. Structural and functional concerns ................................................ 41
Figure 31. Configuration management ............................................................ 42
Figure 32. Node state transitions .................................................................... 43
Figure 33. Modified state transition diagram .................................................... 46
Figure 34. Original BSet ............................................................................... 47
Figure 35. Extended BSet with nested sub-transaction ..................................... 47
Figure 36. Extended BSet with no further sub-transaction ............................... 48
Figure 37. Two simultaneous requests from inside and outside the BSet ........... 48
Figure 38. Extended BSet with blocked request ............................................. 49
Figure 39. Unblocked request ........................................................................ 49
Figure 40. Safe state is reached when all original BSet nodes are blocked ......... 50
Figure 41. Reconfiguration algorithm for Bidan's approach ............................. 52
Figure 42. Performance results ...................................................................... 52
Figure 43. Object chassis and engines.................................................. 53
Figure 44. Mapping functions .............................................................. 54
Figure 45. Dynamic request upgrade.................................................... 54
Figure 46. Dynamic client upgrade........................................................ 54
Figure 47. A mapping operator acts as a binding object......................... 55
Figure 48. Mapping functions .............................................................. 55
Figure 49. Mapping operator. The mapping operator provides an old interface... 56
Figure 50. Mapping functions .............................................................. 57
Figure 51. Upgrade server and nodes..................................................... 59
Figure 52. Chain of future and chain of past simulation objects in a node........ 60
Figure 53. The transform function can use a future-SO’s state to create the new persistent state.......................................................... 60
Figure 54. Overview of the Eternal System............................................ 62
Figure 55. Active replication................................................................. 63
Figure 56. Passive replication............................................................... 64
Figure 57. Evolution Manager (Preparer and Upgrader)......................... 65
Figure 58. Classes involved in interface preserving evolution.................. 66
Figure 59. Processing states during an interface preserving evolution............ 67
Figure 60. Classes involved in interface changing evolution..................... 67
Figure 61. Processing states during an interface changing evolution............. 69
Figure 62. Addition of an object............................................................ 70
Figure 63. Replacement of an object...................................................... 70
Figure 64. Removal of an object............................................................ 70
Figure 65. Relocation of an object........................................................ 70
Figure 66. Classification tree for reconfiguration approaches.................... 71
Figure 67. Composite reconfiguration step............................................ 72
Figure 68. Simple reconfiguration step. Requests that should not be queued.... 73
Figure 69. Unqueued requests issued by an affected object....................... 73
Figure 70. Unqueued nested requests issued by an affected object............. 74
Figure 71. Several organizations with very similar goals needing to coordinate and cooperate with each other without a centralized system................. 76
Figure 72. Centralized system of an alliance approach............................ 77
Figure 73. Archipelago of software islands within the organization............. 79
Figure 74. Archipelago of software islands in the cross-organizational context... 79
Figure 75. Cross-organizational conversation........................................ 82
Figure 76. Global cross-organizational business process.......................... 83
Figure 77. Organizations as a distributed system.................................... 87
Figure 78. Distributed systems vs. cross-organizational interactions............. 88
Figure 79. Simple service.................................................................. 96
Figure 80. Service termination warning and termination period............... 97
Figure 81. Workload and resources....................................................... 97
Figure 82. Training and adaptation costs.............................................. 99
Figure 83. Training and adaptation costs.............................................. 100
Figure 84. Training and adaptation costs.............................................. 100
Figure 85. Internal dependencies of an organization............................. 103
Figure 130. Bias of having same estimated evolution times ........................................161
Figure 131. Implied probability distribution between present and time t ..............161
Figure 132. Uniform to implied distribution mapping ............................................162
Figure 133. Pimplied randomly estimated evolution times ..................................162
Figure 134. Dependent tasks ................................................................................164
Figure 135. Tasks costs .........................................................................................165
Figure 136. Task ordering without considering costs .............................................165
Figure 137. Tasks ordering considering costs ........................................................166
Figure 138. Application and provided service unavailability costs .......................167
Figure 139. Service unavailability cost .................................................................168
Figure 140. Organization and service setup ..........................................................170
Figure 141. Tardiness after ordering ......................................................................170
Figure 142. Unavailability cost without handling multi-counting of costs .............171
Figure 143. Unavailability cost when handling multi-counting of costs ...............171
Figure 144. Organization's dependency graph ......................................................175
Figure 145. Losses originate almost at any time of the foreseeable future ..........194
Figure 146. Losses originate only at 2 moments in the foreseeable future ............208
Figure 147. SOAP message request example .....................................................229
Figure 148. SOAP message response example ...................................................229
1 Introduction

During the last decades, organizations have embraced the path towards outsourcing [1] and vertical specialization ([2], [3], [4], [5]). Organizations have outsourced part of their business processes. By doing this, they increased their productivity and their earnings, but they also became exposed to disruptions of their own activities because of their dependencies on these service providers. These disruptions can be broadly categorized into two types.

The first one regroups all disruptions that occur when the service providers are not able to deliver the services and/or products in the contractually agreed time, quantity and quality. These disruptions are clearly identifiable because their cause originates entirely from the service providers used by the outsourcing organizations. Organizations usually hedge this risk by using multiple service providers or product suppliers [6]. This also induces competition between the different service providers.

The second type of disruptions is more difficult to identify at a glance. It regroups the disruptions faced by outsourcing organizations because they are not able to adapt in a timely manner their workflows to the evolving ones of their service providers. The cause of these disruptions is shared between the service providers and the organizations using the outsourced services. This disruption risk is not usually identified immediately because the outsourcing process is of progressive nature. Disruptions of this kind become more and more frequent as the organization outsources more and more processes to service providers. Thus, by outsourcing part of their business processes with several business partners, organizations may face disruptions of their operations if they are not able to adapt their workflows in time to their changing environment.

While this has been observed for business processes, it is also true for information processes. Thus, the rapid adoption of high-speed corporate networks has led to the explosion of the number of interconnections between applications. This also has made it possible to build single conceptual applications spanning over several machines. The growth of these distributed applications capabilities has been correlated to the growth of their complexity. The early distributed applications were often strongly dependent on the specific underlying platform for which they were designed to run. These applications were very dependent on the technology they had been built with. All elements of a distributed application used to be built with the same technology. Nevertheless, a fierce competition quickly erupted between the main leading information technology actors, each of them trying to impose its own technology stack. To add to the confusion, a continuous flow of start-ups proposed relentlessly new technologies and standards that challenged the dominance of the historical players. Consequently, many different technologies
have been developed to build these distributed applications. To reduce the fast-growing number of technologies used in distributed computing, organizations started coordinated efforts to produce some type of technological standardization for distributed computing. This should have led eventually to improvements in the way distributed applications are conceived, designed, built and maintained. Nonetheless, each organization often chose its specific technology stack and stuck with it for its subsequent new developments.

With the emergence of the Internet, organizations began to want to interconnect their applications not only within the enterprise but also with those of other organizations. Incompatible technologies needed to be bound to work together. Businesses wanted to streamline their data interactions in the most automatic way possible. This led to the creation of inter-application communication methods that became more and more technology agnostic. As of today, the most recent generation of these iterations has materialized in the so-called Service Oriented Architecture (SOA).

The pursued goal of organizations is to integrate their respective processes and to automate the exchange of information that is needed for this integration.

Thus, an organization not only offers information services to, but also uses those offered by other organizations. This means that an organization depends on services it is consuming, but also that other organizations depend on the services it is providing. These dependency relationships between organizations can be represented as a dependency graph as shown in Figure 1.

Since organizations can offer or consume services, at the global level, the graph is not of any particular kind. An organization A can be dependent on an organization B and at the same time, organization B can be dependent directly or indirectly on the organization A.
At the level of an organization, the dependency relationships are more numerous. An organization can have some dependency to another organization because it has integrated one of its business processes with the other organization. The integration of that particular business process necessitates several distinctive interactions between both organizations. Each of these interactions will be materialized through some type of data exchange. All involved applications of the organization using the services offered by the other organization will need to be adapted for each of these interactions. As shown in Figure 2, the more an organization integrates its business processes with its partners, the more it will use their services and the more uncoordinated maintenance work it will have to endure.

![Figure 2. Dependencies of a single organization](image)

An interaction or service offered by the service provider is usually intended to be used by several service consumers. As the business of companies changes with time, so do their business processes. This is also often reflected in the interactions they have with their business partners. Therefore, the services that are offered by a service provider change with time. These changes can originate from the service provider's own internal evolution or because a new functionality has been requested by a consumer of one of its services. These changes to the services offered by a provider will trigger maintenance tasks for all consumers of the
evolving service.

As shown in Figure 3, a changing service will evolve from its current version to a new version. It may also arise that a service provider simply stops offering a particular service. In that case, an organization will have to find an alternate service provider offering an equivalent service to the one currently used. The organization will also need to adapt its programs to use this new alternate service.

Therefore, the more an organization uses services from other organizations, the more it is subject to unplanned maintenance triggered by its service providers. Service providers can change or phase-out their services within the contractual boundaries they have with their respective clients. Because service providers are not aware of the dependencies of their clients towards services of other service providers, they will not take into account any planning conflicts that could arise within their clients' organizations. Therefore, service providers make changes to their services at any time.

In case an organization uses services of multiple service providers, it will receive notifications of changes for its consumed services from these different sources. Since changes to services from multiple service providers are uncoordinated, an organization may be forced to adapt several services simultaneously or within a very close time interval. Each of these notifications will generate some adaptation workload to the applications of the organizations consuming these services. The more notifications being received, the more adaptations must be performed.
Depending on the number of received simultaneous notifications, an organization may have or may not have enough resources to adapt its applications within the respective contractual timeframe of each consumed service as shown in Figure 4.

Thus, there is a probability that service providers change their respective services at moments in such a way that an organization may not be able to adapt its applications while respecting the services' contractual timeframes. In that case, the organization will suffer malfunctions in its own applications. Moreover, if an organization is not capable of adapting itself in time, it may also expose other organizations to disruptions. These chained disruptions arise if an organization consuming services provides itself other services to organizations. A non-adaptation to the evolving consumed services will cause disruptions to the organization's applications. Any service offered by the organization and relying on these disrupted applications would become unavailable as shown in Figure 5. Thus, an organization will not be able to provide its services and this will lead to contractual breaches with its own clients.
The goal of this thesis is to identify if this type of disruptions due to non-adaptation in contractual times can be avoided and if yes, through which means this could be done.

A set of cross-organizational interactions between the IT systems of many enterprises can be easily compared to a huge distributed system. At this stage, a distributed system can be broadly defined as a set of autonomous computing processes that interact with each other to achieve a common goal. As shown in Figure 6, these processes interact with each other within the same host or between several hosts using a data network. Since distributed systems also face upgrade challenges, there might be some patterns or techniques that have efficiently solved them. In the domain of distributed systems, dynamic reconfiguration designates the ways of performing changes to a distributed system while it is running. Because of the analogy between distributed systems and cross-organizational interactions, there might be some type of dynamic reconfiguration solutions for distributed systems that could be applied to a cross-organizational context. Thus, an overview of distributed systems and related technologies will be presented. This will be followed by a detailed exploration of the existing dynamic reconfiguration technologies for distributed systems.
Managing evolution risk of cross-organizational services

Figure 6. Distributed system

After dynamic reconfiguration techniques of distributed systems, cross-organizational interactions will then be presented. This will permit to list the similarities and differences between distributed systems and cross-organizational interactions. The reasons why the dynamic reconfiguration technologies used in distributed systems are not applicable in a cross-organizational context will be discussed. Moreover, I will explain why there will never be an implementation of an automatic cross-organizational upgrade mechanism.

In the light of this first finding, I will propose an alternative approach to minimize the disruptions of the services offered by an organization which are caused by the non-adaptation of the organization's applications to the evolution of its consumed services within its contractual timeframes. This approach will show, under the condition of certain prerequisites, how to predict these future disruptions. It will also permit to analyze the cause of the future disruptions. It will quantify the risk for these disruptions to occur. This whole approach consists of a model that provides a series of quantitative indicators. These indicators help to take decisions to reduce the disruption time induced by the uncoordinated evolution of the different service providers. As mentioned, for the proposed model to be applicable, there are some prerequisites that need to be met.

First of all, the model necessitates that a service version is never abruptly switched to another version at a certain point in time. At least two versions of a service must always overlap with each other as shown in Figure 7. Although this is a prerequisite for the model, it is not specific to it. This is an increasingly important feature in a service world. Particularly in the cross-organizational context, there is a need for services to be offered in multiple versions simultaneously. This need will be discussed and illustrated through the
undesirable situations that could occur if simultaneous versions are not available. I will show why it is essential that two versions of a same service coexist at the same time and that this is the case not only for the model, but more generally for a world of integrated processes.

![Figure 7. Multiple service versions simultaneously](image)

Next, the services that an organization consumes from service providers must all be ruled by contracts. The terms of these service contracts specify the relationships between the parties for each service. The range of the scope covered by these contracts can vary greatly. It can contain the specification on the semantics of the service, the way to access it, the expected service quality, the obligations of both parties, etc. But for the model to work, the contracts must contain at least two elements: the starting date of the service, and the termination delay. As shown in Figure 8, the termination delay is the time the organization consuming the service has between the moment when it is *informed* that the service version will be terminated, and the moment that this service version *is* effectively terminated. When a service provider decides to make any change to an offered service, he will inform the organizations using the service version that the current version will soon be phased out. The service provider’s action of informing the organizations using the service is called the *termination warning*.

![Figure 8. Termination delay](image)

Further, the organization must also build the dependency graph that includes the
services it consumes, its own applications and the services it offers to other organizations as shown in Figure 9. For this, an organization must formally identify the dependency relationships between all these elements.

![Application dependency graph](image.png)

**Figure 9. Application dependency graph**

Then, an organization must also specify the cost that it will endure for the unavailability of any of the elements of the dependency graph. This cost is expressed as an amount in a currency per day of unavailability.

Finally, for the model to be operational, the service providers must provide the *expected timing* for the planned updates to their services. Since the timing of these changes are rarely exactly known even by the service provider, the latter is only bound to provide a probability distribution for the next change of a service as shown in Figure 10. The probability distribution of the service version termination event must be provided on a regular basis, to the best knowledge of the service provider. The gathering of this information will be discussed in detail.

![Service termination event probability distribution](image.png)

*This service version termination event has a probability of happening through time in the future. This probability distribution is given by the service provider and is based on his best belief of his planned schedule.*

**Figure 10. Service termination event probability distribution**
If all these prerequisites are met, the model uses the probability distributions of all services to calculate a series of predictive indicators. The way these indicators are calculated will be described in detail.

The predictive indicators that are calculated include:

- The probability for an organization to not be able to adapt within its contractual timeframes to the new versions of the services it consumes from other service providers.
- The probable cost that an organization will have to bear due to the non-adaptation to new versions of the services it consumes from service providers.
- The identification within an organization of the consumed services that will probably cost the most due to non-adaptation capacity.
- The identification of the services an organization consumes that will probably be the most unavailable due to their non-adaptation to newer versions within the organization's contractual timeframes.

The model itself is an approximation of how maintenance deadlines are managed. But this choice permits to reduce the considered problem to an already known class of problems, the so-called single machine job-shop scheduling class of problems. Since this is known to be an NP-complete problem, I will show that in this very particular context with the inclusion of probability distributions, it is still possible to perform partial computations that yield significant results in finite time.

Using the previously mentioned indicators, it is then possible to determine if an organization allocates too many or too few of its resources to the maintenance of its applications that consume information services from other organizations. It is possible to determine if this over- or under- allocation is structural or if it is only due to the result of an unfortunate conjunction of events triggered by uncoordinated service providers.

The model permits also an organization to price the value of an extension of the contractual termination period length of a particular service. This can help the organization choose to either allocate more resources to maintenance, or to renegotiate the termination period, or even to take the risk to be hit by a probable cost in the future. The organization's choice will depend on the risk it is willing or able to accept and on the difference in cost between its options. Finally, I will describe how this model permits to shift the ownership of critical information from the individuals working for an organization to the organization itself. Thus, the dependency of the organization to some of its key people will be reduced.
Thus, the rest of the document is structured as follows:

- **Chapter 2**: *Introduction to distributed computing*. A brief introduction to distributed computing, its origins, interactions, the different types of middleware and the most commonly used technologies.

- **Chapter 3**: *Dynamic reconfiguration*. An overview of dynamic reconfiguration and an exploration of existing systems and technologies that implement it.

- **Chapter 4**: *Cross-organizational interactions*. A description of cross-organizational services.

- **Chapter 5**: *Dynamic reconfiguration in a cross-organizational context*. An analysis to determine if the dynamic reconfiguration techniques are directly applicable to cross-organizational interactions.

- **Chapter 6**: *Problem description and model proposal*. A description of the underlying problem to be solved and a proposal of a model which aims to minimize disruptions due to the evolution of the outside world of an organization.

- **Chapter 7**: *Definitions, concepts, notions and prerequisites*. A series of necessary definitions and a discussion of all prerequisites of the model and their implications.

- **Chapter 8**: *Model*. The description of the model itself, how it works, and under which conditions.

- **Chapter 9**: *Simulated case*. A presentation of the majors findings resulting of this model.

- **Chapter 10**: *Limitations and weaknesses*. A discussion on the model's limitations.

- **Chapter 11**: *Contribution, discussion and conclusion*. The conclusion and wrap-up.
2 Introduction to distributed computing

2.1 Definitions

Definition - Distributed computing

Distributed computing [7] can be broadly described as threads of computation that spawn over multiple independent processing units and which communicate with each other synchronously or asynchronously to achieve their goals.

Definition - Thread of computations

A thread of computations is a path of code execution through a program. Each thread usually has its own memory stack for local variables, its own program counter which points to the current instruction, and its own life cycle. In common operating systems, there can be many concurrent threads within the same process. Thus, there can be several processes running on a computer and within these processes there can be several threads running.

Definition - Distributed system

A distributed system is a logical aggregate view of a set of independent concurrent processes that are executed on different hosts. These independent yet logically interdependent processes must communicate and interact with each other and must therefore coordinate themselves [7].

2.2 System types

It is possible to relate the types of systems to some other attributes. To do this, we first break down the universe of systems into these categories ([8], [9]):

- Monolithic systems
- Client-server systems
- N-tier systems
- Service-based systems

Monolithic, client-server and N-Tier are intra-organizational systems. They are used within the boundaries of a single organization. Service-based systems are
cross-organizational systems. They usually spawn across multiple organizations\(^1\).

We relate then these categories to four characteristics: the data formats used within the system nodes, the communication protocols used to transfer data from one node to another, the scalability of the whole system and the number of nodes usually in the system\([8]\). The summary relating the different types of systems and the described characteristics is shown in Figure 11.

<table>
<thead>
<tr>
<th></th>
<th>Intra-Organizational</th>
<th>Cross-Organizational</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Monolithic</td>
<td>Client-Server</td>
</tr>
<tr>
<td><strong>Data Formats</strong></td>
<td>Proprietary</td>
<td>Proprietary</td>
</tr>
<tr>
<td><strong>Communication Protocols</strong></td>
<td>Proprietary</td>
<td>Proprietary</td>
</tr>
<tr>
<td><strong>Scalability</strong></td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td><strong>Number of Nodes</strong></td>
<td>Very Small</td>
<td>Small</td>
</tr>
</tbody>
</table>

Figure 11. System types

The Monolithic type of systems is historically the oldest one. Everything is centralized within the same application, whether it is raw computations or user interface. Even though applications may have user concurrency, they are usually not suited for distributed computing. Older mainframe applications fit into this category. They use mainly proprietary data formats and communication protocols. They are not scalable beyond a certain limit\(^2\), and there is usually a single computing node involved\(^3\).

The Client-Server type of systems comprises the systems where the computations are split into two distinctive processing units: the server, which usually processes all shared data-related tasks, and the client, which handles processing that is specific to a single interaction session. This type of systems also uses proprietary data transfer formats and communication protocols. It is scalable in the sense that some of the computations can be deported to the client processing unit, rather than centralizing all computations in the server. The scalability is in fact bound to the amount of computations that the server can handle, i.e. to the scalability of the server node. The number of nodes involved in this type of system can spawn from two nodes to a few hundred nodes\(^4\).

The N-tier type of systems is an extension of the client server type. In this type,

---

1 They are also used within a single organizational context.
2 The limit is the maximum upgrade of the hardware, usually of the same vendor.
3 Some mainframe systems offer the possibility to combine several hardware boxes, but the system is still considered as a single node.
4 \(N\) clients + 1 server.
the server node itself is split into several nodes. The data formats are usually more open than the monolithic and the client-server system types. The scalability of this type of systems is usually better than the monolithic and the client-server types because the workload of the server computations can be distributed onto several systems. The number of nodes can be more important than in the client-server type since the server part is more scalable. Thus, the system can accept more clients.

The Service-based type of systems is a type of systems where the computations are distributed across several independently managed systems. They use standard data formats and communication protocols. Their scalability is usually high and the number of nodes within the distributed system can be more than several millions.

2.3 Client and server

A client is defined as a requester of services and a server is defined as the provider of services [10]. A process, the client, will ask for the execution of some code residing in another process, the server. This process often runs on a different computer than the client. The client process will optionally furnish some data as parameters to the server process which will execute the required computations and return the results (if there are some) to the client. A single process can be simultaneously both a client of some other process and a server for other ones. The term client-server usually means that the client is not only a passive terminal but that it has its own processing path. The client-server architecture is therefore at the hart of network computing. A client-server architecture can use many communication patterns like synchronous TCP sockets or an asynchronous message-passing system to implement these request/response actions. As shown in Figure 12, the major architectures in the client-server paradigm are two-tier and three-tier architectures. Two-tier architectures connect directly a client to a server without any other intermediary. Multi-tier architectures are composed of several layers between the client and the server and will handle and route the request to its destination. In this latter case, the software layer (or layers) between the client and the server is called middleware [10]. To handle contextual persistence between several client-server interactions, the client usually initiates a session on a server and then uses this session as context for all subsequent requests.

5 Service-based systems are often of cross-organizational nature. To the opposite, the Monolithic, Client-Server and N-Tier types of systems are usually of intra-organizational nature.
6 A TCP socket is an endpoint for point-to-point communication done with the Transport Control Protocol (TCP).
7 Also known as “multi-tier architectures”, even though multi-tier architectures can have more than three tiers.
A client can be viewed as a triggering process. On the opposite, a server can be viewed as a reactive process [11]. A client initiates an interaction between itself and a server. In synchronous interactions, a client that initiated a request waits for its completion by the server before continuing its own execution. A server may possess its own autonomous computation paths, but usually the server waits for requests from the clients and then processes the required services. A server is usually a non-terminating process and often provides service to more than one client [12].

Functional segregation is an important part in a client-server architecture. The choice of the client and server functionalities will define the degree of scalability and asynchronous evolution possibilities. There are no specifications on how this segregation should be operated. In a simple client-server system with a user interface, the most common segregation is to move to the client all the user interface-related functionalities and to the server all the application logic functionalities [12]. They may then evolve independently if the interfaces between both parts don’t change.

### 2.4 Interactions

Distributed computing is, in essence, a way of splitting computations and data into defined communicating parts. The interactions and communications between these parts compose a logical skeleton for a greater global application. Splitting computations and data into independent smaller pieces, and binding them together with interaction and communication technology, is in fact a compositional approach.

It is common belief that the development of complex large systems can be eased by using compositional approaches [7]. As new technologies and design concepts appear over time, they are usually stacked onto the existing layers to maintain
backward compatibility. These new concepts are often more abstract than the underlying software layers on which they rely. However, it would be extremely valuable if the relationship between the highest and the lowest layers would be correctly identified and modeled, which is currently not the case [7].

The interactions between the different collaborating software elements of a distributed system can be either synchronous or asynchronous [13]. Each of these communication patterns are more or less suited to the functionalities of the distributed system. They are rather complementary approaches than competing approaches.

2.4.1 Synchronous interactions
A synchronous interaction is defined as the situation where a thread of computations in a client application initiates an interaction with a server application and waits for the completion of that interaction. The interaction comprises the communication from the client application to the server application, the execution of the request by the server application and the communication of the result from the server to the client as shown in Figure 13. The completion of the interaction has different meanings whether it is considered from the client side or from the server side. From the client side, the completion of the interaction can either be an applicative response, where this response is the result of the server application, implying that both communications from the client to the server and back from the server to the client were correctly handled, or the completion of the interaction can be an information of communication failure that is passed to the client application thread by the local communication layer.
From the client side, the communication failure can be either the impossibility of sending the request, or the timeout of a response from the server. In the latter case, the interaction is not complete from the client side but may be from the server side. This case happens when an interaction initiated by a client reaches the server application which handles correctly the request, processes the computations, sends back the results to the client where an error arises. After the communication layers have handled correctly their task, the server usually gets the information that everything has worked fine and therefore moves on to serve another request. But on the client side, after the communication layer got the result from the server, the application layer can still produce an error or timeout. In this case as shown in Figure 14, the server can not have the information that the client has had an error and from the server’s point of view the interaction was accomplished without an error.

**Figure 13. Synchronous interactions**
A problem occurs at this point. From the client side the interaction has not completed correctly but from the server side it has.

Figure 14. Synchronous uncompleted interaction

Thus, depending on the protocol between the client and the server, an interaction can be considered to be not completed on the client side but completed on the server side.
2.4.2 Asynchronous interactions
An asynchronous interaction is characterized by the situation where a thread of computations in a client application initiates an interaction with a server application but does not wait for the interaction completion to continue its own execution. There might be a result produced by the server application that is sent back to the client application. If this is the case, the client will act itself as a server and handle the server's response as shown in Figure 15. In this type of interaction, the sender handles usually only communication failures for the outgoing requests to the server. The processing errors that happen on the server side are not known by the client unless the server sends back explicitly this information to the client\textsuperscript{8}. Thus, communication failure handling in asynchronous interactions are only one-sided. A Message-Oriented architecture relies usually on this type of interactions.

![Figure 15. Asynchronous interactions](image)

2.5 Workflows
Complex process management is usually driven by workflow management systems \cite{14} which allow the specification, execution and monitoring of workflows. Workflow management systems have been extended from their initial intra-organizational goals to handle the increasing number of cross-organizational processes. Each workflow management system uses some specification language to define workflow types. A workflow type defines all the states, transitions and tasks involved in the definition of a workflow process. A workflow instance of a specific workflow type is created each time the process is initiated.

Outsourced processes can also vary greatly in their complexity. Some of them will consist of a simple request-response interaction while the more sophisticated ones will be composed of many interaction steps. The number of participants in an workflow can also be greater than the sole service provider and requester. A

\textsuperscript{8} The client can't rely on a connection loss for implementing an error handling mechanism.
provider of a complex service can itself outsource part of the service to sub-providers. In this case, the completion of a service can involve an arbitrary number of third-parties. Commonly encountered workflow patterns have been classified in [15]. The increasing dynamic nature of cross-organizational workflows is leading to constant workflow type evolution. Because of this, dynamic workflow schema evolution has been intensively researched ([16], [17], [18], [19], [20], [21], [22], [23]) with the main focus of the papers targeted to the adaptation of ongoing workflow instances to newly evolved workflow types. There has been less attention towards the evolution of the integration code in client environments of evolving cross-organizational workflows. When cross-organizational workflows evolve, the integration code in the client environments can be heavily affected. The integration of these client parts into the workflow applications can be viewed as a distributed application. Because dynamic reconfiguration of distributed applications have been a topic of research since the mid 80's, they may provide some solutions to the problem of automatic dynamic reconfiguration of cross-organizational workflows. Moreover they could also provide some help for the task of managing the integration code evolution in client environments with changing cross-organizational workflows.

### 2.6 Distributed system architectures

The creation of distributed systems has spurred many tentatives to standardize the way they should be built. Several architecture patterns have emerged over the years. Middleware [24] is a generic term that designates the class of software that allow multiple applications or application parts to interact with each other across a network as shown in Figure 16.

![Diagram](attachment://Figure_16.png)

**Figure 16. Middleware layer**
Thus, Middleware offers applications a common way of interacting which is often independent of the technologies and languages with which these applications have been built.

There are many different styles of middleware that have been developed, as shown in Figure 17. The most frequently encountered styles of middleware are transactional, procedural, message-oriented, object or component middleware and service middleware. Among the best known distributed computing architectures are the Remote Procedure Call (RPC) ([25], [26]), the Client-Server Architecture [27], the Multi-Tier Architecture, the Distributed Object Architecture (DOA) ([28], [29], [30]), the Message Oriented Architecture (MOA) ([31], [32]) and the Service Oriented Architecture (SOA) ([33], [34], [35], [36]). These architectures are not equivalent in functional scope and therefore, they often overlap. In large distributed systems, these architectures are frequently used together [37] because of the fragmentation of the distributed system elements and the heterogeneity of the technologies used throughout the whole distributed system.

![Figure 17. Embedded and stand-alone middleware](image)

### 2.6.1 Transactional Middleware

Transactional middleware consists of a stand-alone coordinating layer of software, often denominated transaction manager, which oversees a transaction involving multiple components on multiple hosts. Server components offer services which are used by client components. A transaction initiated by a client component can encompass multiple server components, even if they reside on different hosts. The transaction manager will insure an all-or-nothing execution of the transaction for all the components involved in the transaction. This is defined as Distributed
Transaction Processing (DTP). For a component to be part of a distributed transaction, it must implement a so-called two-phase commit transaction protocol. For interoperability reasons, the industry has adopted the two-phase XA-Protocol of the X/Open Group, a worldwide organization supported by almost all the major IT companies. The XA-Protocol, also called XA interface, is a bidirectional interface between a transaction manager and a resource manager. XA offers programmatic interfaces to the Distributed Transaction Processing (DTP) model. XA is widely supported by almost all major database management systems and application servers.

2.6.2 Procedural middleware

Procedural middleware essentially includes the Remote Procedure Call (RPC) ([25], [26]). Remote Procedure Call (RPC) is a client-server infrastructure that reduces the complexity of developing applications that span multiple processes on multiple operating systems and network protocols by hiding from the application developer the details of the various programmatic interfaces used to perform client-server calls. RPC is a synchronous communication model [38]. The original idea of RPC started in the mid 70's. The first full RPC implementation was made by Birrell and Nelson in 1984 [25]. Large IT industry companies like Sun [26] rapidly endorsed RPC which led to commercial implementations.

RPC has a scalability and fault-tolerance limited to its built-in features. In a client-server application using RPC, a client accesses the server through specials functions named Remote Procedure Calls. These function calls will hide the underlying communication needed to pass the calls to the server. The links and dependencies between the client part and the server part are defined with propriatory languages and tools and are built at compile time. They are then incorporated in the client and server part of an application. The functions are usually defined with a custom Interface Definition Language (IDL), and are then compiled. The result of the IDL compilation are two pieces of code, a stub for the client and a stub for the server. These stubs are invoked when the application requires a remote function. As shown in Figure 18, these stubs are then linked with the corresponding application part, either with the client or the server. These stubs handle communication and marshalling/unmarshalling\(^9\) of the transmitted data.

---

\(^9\) Marshalling stands for transforming in-memory data into a transportable format. Unmarshalling designates the opposite, i.e. recreating in memory a data structure represented in a transportable format.
RPC is a synchronous middleware and is not reflexive since the stubs are created at compile time. This means that the function interface cannot be changed at run-time. RPC is not a discreet middleware layer as are Message-Oriented Middleware (MOM) or Distributed Objects. As shown in Figure 17, RPC is an embedded form of middleware which reduces evolution and reconfiguration flexibility.

The control of the whole program execution passes from the client process to the server process. The client process blocks until it gets the answer from the server. The server process receives the request, executes the requested service and sends back the corresponding result. The client process receives the result and continues its execution as shown in Figure 19. The stubs take care of the transformation of the arguments and results into a proprietary format that is used during communication.

10 Message-Oriented Middleware (MOM) and Distributed Objects are described in the following sections.
2.6.2.1 RPC advantages
It is mentioned that RPC is easier to understand for the average developer than a message passing system [40]. The semantics of a remote procedure call are familiar to all developers ([40], [41], [42]). The complexity of the development of distributed processing is reduced with RPC because the location of a function is not shown in the client and the server code. RPC allows the remote component to be accessed without knowledge of the network address or any other lower-level information. It is possible to access a function in a remote machine in the same way as in the local program without having the semantics of the client code changed.

2.6.2.2 RPC shortcomings
Because of the synchronous nature of RPC, an application that uses it can only function correctly when both client and server are running at the same time. On the contrary, an asynchronous message-based distributed system will be able to function even if one of the system's participants is unavailable. This means that from a production view, a client application using RPC is in fact comprised of the client part and of the server part. It is also recognized that RPC is not well-suited for applications involving distributed objects or object-oriented programming [38]. Furthermore, RPC implementations are proprietary. Therefore, different RPC solutions are not interoperable. Moreover, there is no standard scalability or failover mechanism defined in RPC. Thus, these features will depend on the chosen RPC implementation. This hurts the common enterprise view of limiting the number of technologies for executing similar tasks like inter-application communication. Because RPC is a technology that is strongly compile-time oriented, features like load-balancing or failover mechanisms are usually externalized into a separate middleware. If an application has been built using RPC, it is also necessary to pay attention to the number of users. Depending of the number of users, RPC may become inefficient. Network congestion and the handling of many different contexts for many clients can quickly reduce the server's throughput. The order of magnitude of these limitations depends on network bandwidth and server power, but is more in the hundreds than in the thousands [38]. Thus, connection multiplexing is frequently used in heavily loaded environments. In case several remote calls are made to fulfill a logical action and if the server maintains some form of state between the calls, there might occur some memory or resource leaks because distributed garbage collection is a challenging problem ([40], [43]). Finally, RPC may not be suited for certain communication patterns. For example, it is not suited to situations where a server response must come back from a different process than the original recipient of the request.
2.6.2.3 RPC extensions
RPC has been extended by several systems that handle certain types of asynchronicities either for concurrency or for frequently disconnected networks [40]. Promises is a RPC framework that deals with the concurrency issue. It allows parallel execution of the calling thread and the remote call and will only block when the value of the remote call is effectively used ([40], [41]). This approach can be viewed as similar to deferred pipelining. "Rover" is a queued RPC system for communication with frequent connection losses. Remote Procedure calls are queued while nodes are disconnected and they may be reordered such that high priority calls get performed first when connections are slow ([40], [44]).

2.6.3 Message-Oriented Middleware (MOM)
Message-Oriented Middleware ([31], [32]) defines a class of software infrastructure that enables loosely coupled and usually asynchronous interactions between independent applications or processes residing on different hosts. The communication between the applications is achieved through the use of messages. These messages are finite data chunks that encapsulate some information. It is to notice that multiple applications can receive the same message. When communicating through a MOM, applications act as clients relatively to a MOM server. As shown in Figure 20, there are three major topologies when linking applications together through Message-Oriented Middleware. These topologies can be either centralized like the Hub and Spoke topology, or distributed like the Snowflake or Bus topologies.

![Figure 20. MOM topologies](image.png)

Message-oriented middleware implements asynchronous interactions. As shown in Figure 21, an application A posts a message into a message queue. The message is then handled by the message routing infrastructure and forwarded to a destination message queue.
At the reception of a message in the destination queue, a signal is dispatched to a listening component and informs application B that a message has arrived and that it is ready for being processed. While application B reads the message and handles it, application A can continue to process other tasks if necessary. If an output of application B must be sent back to application A, another message is sent back from application B to application A.

There are several ways of exchanging information using messages. These different communication patterns have created some sub-families [32] within the Message Oriented Middleware itself. Among these MOM sub-families, three of them have emerged: Point-to-Point MOM, Request-Reply MOM and Publish-and-Subscribe MOM.

### 2.6.3.1 MOM advantages

Message-Oriented Middleware has some advantages relative to the other types of middleware. One of its main advantages is loose coupling between a client sending a message and a server receiving it. It is asynchronous and can handle multicasting. It can implement failover capabilities and can be used in very heterogeneous environments. The communication resources can be shared more efficiently than with other interaction patterns like RPC. Message-oriented middleware helps build more scalable distributed systems because of the removal of the synchronous communication latency on the client side. At the opposite, the implementation of a synchronous interaction model with a message-oriented middleware is more complex and less efficient than with procedural middleware [45] since the model must be implemented at the application level on the client side. The software layer for the message handling is independent from a participant. There are many commercial products available. They include IBM’s MQ Series and Sun’s Java Message Queue (JMS) ([46], [47]).
2.6.3.2 MOM shortcomings
MOM possesses a relatively low-level of abstraction. There are many products on the market and they don’t implement all interaction patterns (Point-to-Point, Publish/Subscribe, Request-Reply). These products are incompatible with each other which means that bridges between them must be built. This has lead to the emergence of middleware islands ([48], [49]) in big organizations in which often several middlewares coexist. To overcome the limitations inherent to this situation, some solutions try to abstract the underlying messaging infrastructure. One of the popular approaches is the Java Message Service (JMS) [46]. Due to the varying types of data that can be transported with MOM, the marshalling is usually done at the application level.

2.6.4 Object-oriented middleware
Object-oriented middleware ([50], [51]) is characterized by the use of the Object-Oriented paradigm in the context of distributed computing. One often refers to as the Distributed Object Architecture (DOA). DOA derives historically from RPC middleware. It consists roughly of an independent software infrastructure that provides a mechanism to be able to locate objects in a remote application and to invoke synchronous methods on these remote objects as if they were local. This infrastructure is called an Object Request Broker (ORB). A distributed object is defined as an object which can be used remotely by other applications. An application that wants to use a distributed object remotely is defined as the client application. A distributed object is represented in a client application by a proxy object as shown in Figure 22.

![Figure 22. Distributed Object Architecture](image)

A proxy is only a representation of the remote object. A proxy does not contain any state or code specific to that object. A proxy is the local interface in a client process to access the distributed object. The distributed object is called, from a client's perspective, the remote object. A proxy contains a reference to the remote object it represents. The methods supported by the proxy correspond to the methods of the remote object. All arguments used in the call of a method on the proxy will be passed to the corresponding method of the remote object as shown in Figure 23.
A proxy object has the same role in the distributed context as an interface in an object-oriented model\textsuperscript{11} of an in-process context as described in [52]. The method call of the remote object through the proxy is similar to a local call. This property is called access transparency \textsuperscript{39}. The transfer of a call from a proxy object to a remote object is achieved through the Object Request Broker. The ORB is responsible for linking the calls of a proxy object to a remote object as shown in Figure 24. It can be basically associated with a distributed object reference manager. The abstraction of a remote object location is called location transparency \textsuperscript{39}.

\textsuperscript{11} "One of the main ideas of object-orientation is information hiding. Objects encapsulate state, and client objects may interact with the object only through a well-defined interface. This interface represents the contract between the clients and the object, imposing a certain structure for the way the state of the object can be accessed and manipulated from the outside. This contract can be strengthened to enhance consistency of an object." [52]
The Object Request Broker is also responsible for providing some special services. One of these services is to provide references to some particular system objects. A client application will use the Object Request Broker to obtain the references to the objects implementing these services. For example, in the case of a distributed object system, the reference to an object is obtained through a special service, the Naming service. This service is itself implemented as an object.

An object is said to be *migratable* or *relocatable* if it can be moved from one object container to another while being used without causing any impact to the global system. The result of an invocation of any of its methods before or after the migration should be exactly identical. Object migration must be transparent to all client applications.

The most frequently used object/component middleware comprises the Common Object Request Broker Architecture (CORBA) ([53], [54]) and Sun’s Remote Method Invocation (RMI) specification.

### 2.6.5 Advantages of the Distributed Object Architecture

Distributed Objects make possible the transparent use of remote objects. A reference to a proxy of a remote object can be used by a developer as if it were a local object. This location abstraction can improve the code readability, but by hiding the remote nature of an object, this can induce some other problems as described in the next section.

### 2.6.6 Shortcomings of the Distributed Object Architecture

The garbage collection of distributed persistent objects ([55], [56]) is a weakness of any DOA approach. Since persistent objects can be created, used and deleted through the network, it is necessary to invalidate a distributed object only when it is not reachable anymore. If it is not possible to determine if a distributed persistent object can be safely deleted, it will stay in the persistent store forever. Some Distributed Object Architectures like CORBA ask for distributed persistent objects to be deleted explicitly by a client. This approach is considered to be error-prone [56]. Especially, cyclical garbage collection is a challenge in large-scale distributed systems. Moreover, a Distributed Object Architecture creates tight links between applications. Any change to an object behavior or interface impacts all systems using this object remotely. DOA models handling this issue will be presented in chapter 3. If all applications that are involved in a distributed system are not managed by a single authority, it is necessary for the teams in charge of the different applications to coordinate before any upgrade. This point will be detailed in section 5.2. Furthermore, security considerations add another layer of complexity to a DOA [57]. Finally, an improperly designed
application based on a DOA can lead to some performance issues. For example, it is imperative to define the correct granularity of distributed objects. A call on a method of a remote object to get a single value is very time consuming due to all the inherent communication overhead that it generates. Thus, designing too fine-grained objects can result in very poor performance. Moreover, if there is total location transparency for an object, the developer may code an algorithm without knowing that the objects and the methods he uses are in fact remote objects which leads to performance impacts. Another performance issue can also arise if objects are persistent and stored in relational databases. In this case, there can be a significant performance gap between querying large collections of persistent objects and raw SQL queries. This performance issue is caused by the additional software layers present in the DOA that are necessary for using remote objects stored in the database in a transparent manner.

2.6.7 Component-oriented middleware
A component ([50], [51]) is a unit of composition for a specific run-time environment. A component is often composed of several object classes. A component is deployable in its target environment independently of other components. It implements some specific interfaces required by the environment and is contextually parametrized at deployment time. As defined in [58], "A component denotes a self-contained entity (black box) that exports functionality to its environment and may also import functionality from its environment using well-defined and open interfaces. In this context, an interface defines the syntax and semantics of the functionality it comprises (i.e., it defines a contract between the environment and the component). Components may support their integration into the surrounding environment by providing mechanisms such as introspection or configuration functionality.". Component-oriented middleware can be associated to a particular form of object-oriented middleware. Thus, they will both be treated as object-oriented middleware in this thesis.
2.6.8 Service middleware

Service middleware or Service-Oriented Architecture (SOA) ([33], [34], [35], [36], [59], [60], [61], [62]) is a type of middleware that induces very loose coupling between interacting applications and is based on the use of services. In an SOA approach, some applications, the service providers, publish sets of services\(^\text{12}\) in a centralized directory or registry, and some applications, the service consumers, use this registry to dynamically discover published services, locate the service provider of a particular service, and then invoke that service from the provider as shown in Figure 25.

![Figure 25. Service oriented middleware](image)

The term "services" is often confusing since there is no single definition for the term. The term has been around since the 70's but has been used more and more as the number of interconnected applications between enterprises increased [33]. In the context of distributed computing, a service has been defined as a discrete, technological independent, network-reachable business function ([59], [60]). A service is not thought to be equivalent to a component even though from a client point of view, the approaches possess conceptual similarities and share also partially their terminology [33]. Nevertheless, some authors claim that a service can be associated to a component with an interface [59]. Because of this ambiguous categorization, it was stated that components are the natural approach to build services, in the same way objects were thought to be the best way for building components [33]. A common view of the conceptual hierarchical relations between services, component and objects is pictured in Figure 26.

\(^{12}\) In the sense of "all necessary information needed for locating and invoking the services."
The term "service" is also generally linked to the idea of a transaction unit described by a contract of high abstraction level. Thus, a service is assumed to be independent of a particular language, platform or distributed framework [60]. In the case of legacy systems, where an interface is offered to other applications through services, it is often meant that the legacy system will provide some of its key logical functions in an encapsulated network-reachable manner. To add to the confusion, the term "services" is often used to designate web services. Thus, the term "service" also suggests the idea of heavy use of XML and a string-based marshalling schema.
Nevertheless, in almost all cases, services are seen more as a process model rather than a data model. Throughout all definitions, a service possesses the following characteristics [35]:

- A service must be dynamically discoverable and invokable
- A service must have a platform-independent interface contract
- A service must be network-interoperable

The loose coupling between a service consumer and a provider suggests that the consumer should not have any information about the provider’s implementation and that the location of the service is determined at run-time. It has also been mentioned that the interface contract for the service shall be platform and language-independent [35]. It is also suggested that there should be multiple service versions available simultaneously, since clients do not necessarily belong to the same organizational unit and cannot therefore synchronize for changes or adaptations [63].

2.6.9 Web Services

The most popular implementation of the Service Oriented Architecture are Web Services ([35], [64], [65], [66], [67], [68], [69], [70], [71]). The term Web Services defines one particular set of technologies that implements the SOA concept. It is important to note that there are many different standards trying to unite the use of web services, but there has not been yet a clear winner since there will be heavy commercial outcomes for the winning consortium. Nevertheless, all Web Services offerings rely heavily on HTTP and XML and are composed of SOAP, WSDL and UDDI ([70], [71]) which are described hereafter. The combination of SOAP, WSDL and UDDI fulfills the Service-Oriented Architecture dynamic discovery and invocation requirements [35] described below.

2.6.9.1 Simple Object Access Protocol (SOAP)

The Simple Object Access Protocol (SOAP) ([64], [65]) defines an XML-based communication protocol for exchanging messages between applications. Although it is independent from any transport protocol, it is often used in conjunction with HTTP. An application, receiving a SOAP message and processing it, is called a SOAP endpoint. A SOAP message is an XML string. SOAP messages rely heavily on XML namespaces. As shown in Figure 27, a SOAP message comprises an envelope, some optional message headers for extensions and the message body.

---

13 HTTP assures interoperability and XML provides the platform-independent interface contract.
14 Dynamically discoverable and invokable, platform-independent interface contract, network interoperable.
The SOAP envelope serves as a container for the other elements of the SOAP message. The SOAP headers are used to provide contextual information of the SOAP request that is not directly related to the message body. For example, a transactional system built on SOAP messages could use SOAP headers to propagate some transaction-related information. This information is not related in any way to the information contained in the SOAP message body. This permits SOAP messages to be extended to fit any specific message processing performed by applications. The SOAP message body contains the application-specific data of the message. Please refer to Appendix A in the appendices for a SOAP message example.

2.6.9.2 Web Service Description Language (WSDL)

The Web Service Description Language (WSDL) ([66],[71]) defines an XML format for accessing XML-based networked services. The description of the XML requests is independent of the protocol used for application communication.

2.6.9.3 Universal Description, Discovery and Integration (UDDI)

Universal Description, Discovery and Integration (UDDI) is a specification for distributed web-based information registries of Web Services ([67], [68], [69], [70], [71]). UDDI is composed of an XML schema for SOAP messages. The UDDI XML schema provides four types of information: business information, service information, service binding information and service specification information. UDDI is meant to be used to locate a particular Web Service or to find what Web Services are offered by a business partner. Once the desired Web Service is found, the Binding Template is retrieved for this service. The Binding
Template contains all information that an application needs to invoke correctly the web service at run-time. A UDDI registry can also be used to implement some forms of dynamic reconfiguration. In case of service execution failure [69], an application should reconnect to the UDDI registry and re-obtain a new version of the Binding Template. If the new version of the Binding Template is not the same than the one used for the failed call then a new request is issued by the application.

2.6.10 Services advantages
Service-Oriented Architecture favors a lot interoperability between heterogeneous platforms [70]. The discovery, location, and binding of a service at run-time simplifies application development by moving some parametrization from the coding phase to the deployment phase15.

2.6.11 Services shortcomings
Security, service composition for complex business transactions and service semantics are considered to be problematic [70]. Since cross-organizational business processes are complex and involve multiple services, complete service choreographies would be more useful than individual service entries in service registries [72]. The SOA vision for global services registries has not lived up to its promises ([73], [74]).

2.7 Technologies comparison
The previously described distributed technologies have overlapping features and functionalities. They also have their strengths and weaknesses depending on the context in which they are used. For a project involving a single software developer, creating a complete service infrastructure just for getting two processes to communicate may be an overblown solution and would not justify the cost of development and maintenance. But if an organization would like to have its applications to interact with the ones of other organizations, it would then preferably choose a technology that induces the least coupling between its own systems and the ones of its partners. Thus, in the same manner system types have been associated to specific characteristics as described in section 2.2, it is possible to do this with distributed computing technologies.

There is a relation between how much of a computation is intended to be publicly available16 and the choice of technology through which this computation is made

---

15 Application code keeps only the “what”, not the “were” or the “how” which are specified at deployment or run-time.

16 In the sense of the granularity of the organizational authority managing the applications that could use the computation remotely. It could be applications from the same team, from the same department, from the same organization, or from other organizations.
available for distributed computing. Because of the heterogeneity of the technological environments of all the clients using some remote computation, the more public the latter is, the more technology-agnostic it must be [75]. As a consequence, the more a distributed computation is public, the more it is necessary to specify its behavior. Moreover, when the processes of organizations start to depend on external services, the way their information systems interact must be specified through formal contracts17.

In the same way, there is also a relation between how much of a computation is public and the granularity ([76], [77]) of the computation itself. There are two reasons for this: the first one is that the latency in distributed computing is not zero. Therefore, too many function calls or requests from an application to a remote system can quickly lead to performance deterioration because of the time spent for communication only. Further, error handling with many service calls is more difficult than with a single coarse-grained one [77]. Although fine-grained granularity can still be suitable in an inter-organizational context, coarse-grained services add the most value in the cross-organizational one [77]. Thus, the more public a computation is, the coarser is its granularity.

All these relations are linked to the type of technologies used in the different contexts of distributed computing. The summary [8] relating the different types of distributed computing technologies and the previous described characteristics are shown in Figure 28.

<table>
<thead>
<tr>
<th>Granularity</th>
<th>Procedural</th>
<th>Objects</th>
<th>Components</th>
<th>Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contract</td>
<td>Defined</td>
<td>Private/Public</td>
<td>Public</td>
<td>Published</td>
</tr>
<tr>
<td>Reusability</td>
<td>Low</td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>Coupling</td>
<td>Tight</td>
<td>Tight</td>
<td>Loose</td>
<td>Very Loose</td>
</tr>
<tr>
<td>Dependencies</td>
<td>Compile-Time</td>
<td>Compile-Time</td>
<td>Compile-Time</td>
<td>Run-Time</td>
</tr>
<tr>
<td>Communication Scope</td>
<td>Intra-Application</td>
<td>Intra-Application</td>
<td>Inter-Application</td>
<td>Cross-Enterprise</td>
</tr>
</tbody>
</table>

**Figure 28. Comparison of distributed technologies** [8]

It is suggested in [59] that current technological solutions are not process-oriented, but more component-oriented. It is also pointed out in [59] that the user interface and the database layer have slowed down the development of process-oriented solutions. It has also been stated that “not every good component transformed into a service makes a good service” [33]. In particular, it may be important to keep service granularity coarse enough to avoid too much communication between applications [34].

17 This will be detailed in sections 4.5 and 7.2.
It is also mentioned in [61] that even if current component or distributed technologies like CORBA or Web Services offer some features like reconfiguration mechanisms, dynamic lookup, loading and binding of components, the built-in level of support and the final costs of these features may vary considerably among the technologies.

Since services are used to provide interfaces for the execution of some more abstract processes, it is common to model the interaction sequences between a service consumer and a service provider with UML use-cases and diagrams. In this case the use of services helps expressing the interaction steps without dealing with the implementation details [61].

### 2.7.1 Middleware Fragmentation

The reason why so many types of middleware exist is that there are many different types of distributed computing situations and that there is no single middleware that is optimal for all of them. The fact that middleware has been developed continuously and the fact that many vendors have proposed proprietary solutions is also a factor for the fragmentation. Many companies have therefore not been able to keep up with the pace of change in middleware technology. Once a company has chosen a middleware solution, the cost of changing it is proportional to the number of applications using it. Each middleware also has a high complexity associated with its use. The use of middleware has also raised new problems as it has solved previous ones. Therefore, some middleware solutions have often been abandoned as quickly as they had been adopted. [78]

### 2.7.2 Middleware shortcomings

Even though many middleware solutions are already widely adopted, it is sometimes thought that they are relatively inflexible systems that adapt with difficulty to changing environments. Middleware solutions are also perceived as not being able to scale beyond local area networks [39], with the notable exception of service middleware\(^\text{18}\).

---

\(^{18}\) Described in section 2.6.8.
2.8 Transactions in distributed systems

Transactions [79] in distributed systems aim to work across multiple systems the same way they do on a single system. The target of distributed transactions is to retain the ACID\(^{19}\) properties across all systems involved in a distributed transaction [80]. Additionally, they must provide global atomicity and global deadlock prevention [79]. Global atomicity guarantees that the individual transactions of each involved system in the distributed transaction will either all commit or all rollback. Global deadlock prevention must ensure that no deadlock stalls the system forever. This issue is usually dealt with through timeouts that are used to rollback the distributed transaction after a while, which triggers a new transaction attempt.

The most widely known and used distributed transaction mechanism is the two-phase commit protocol. Distributed transactions would work if all systems involved in a distributed transaction implement the two-phase commit protocol and if all individual transactions run in the serializable mode. However, this is often not the case [80]. Performance is the most cited reason for not using the serializable mode in an inter-organizational context. Indeed, using a two-phase commit protocol without all participants following the rules can lead to many detrimental situations. This comprises global deadlocks, data corruption or loss, unexpected timeouts, partial rollbacks and even incorrect results with consistent individual transactions.

There are even more reasons that have prevented distributed transactions to be used on a wider scale, especially in the cross-organizational context. One of these reasons is the high complexity of implementing distributed transactions across multiple systems. Another one is the fact that a global deadlock can happen even if all the involved systems in the distributed transaction are working properly.

Thus, it is believed that distributed transactions are almost not used anymore [81] in large distributed systems, especially if the context is cross-organizational. Rather, each involved system uses local transactions in a cross-organizational business process. In this scheme, there is no single global transaction that commits or rolls back all changes atomically. Instead, compensating transaction mechanisms [79] are used to cancel changes if necessary.

\(^{19}\) Atomicity, Consistency, Isolation and Durability
3 Dynamic reconfiguration

A common scheme in distributed computing is that a set software elements residing in different processes on different hosts perform their tasks and communicate with each other. Some of these elements will act as server components, or as client components, or even as both server and client components. When the software elements communicating together form a single logical application, the distributed system is called a distributed application. The distributed application is usually under the control of a single authority that will decide when and how to upgrade the different distributed parts of the application. The term reconfiguration is often used to describe a modification on the distributed application. Reconfiguration can be divided into two distinct categories: topological reconfiguration, which handles the location of the software components composing the distributed application, and functional reconfiguration, which handles the evolution of what the distributed application actually does or how it does it. The process of reconfiguring a distributed application while it is running is called dynamic reconfiguration\[82\].

The different software elements of a distributed application are often independent regarding their capabilities but interdependent regarding the services they offer to each other. Because of this independence, each element evolves independently and regardless of the evolution of the other elements. However, the interdependence of an element with other elements it is depending on induces certain constraints upon it. As long as changes or evolutions occur inside a defined element, and as long as these changes are not perceived by the other ones, the entire conceptual system will remain in a consistent state. But, when the changes are sufficiently important to modify the way the different parts interact, there must be a form of change synchronization such that the whole system stays in a consistent processing state.

Applying changes to an element of a distributed application will therefore trigger the need for a reconfiguration of the involved software elements. As this reconfiguration shall not stop the execution of the entire system comprised of all the parts, it shall be done dynamically, ideally without any unavailability or malfunction of the entire distributed system. This reconfiguration shall also ensure that the ongoing running computations are either completely executed or aborted such that the state of the global system stays consistent.

---

20 As defined in [82]: "Dynamic reconfiguration of a distributed application is the act of changing the configuration of the application as it executes. Examples of configuration changes are replacing a module, moving a module to another machine, and adding or removing modules from the application."
3.1 State of the art in dynamic reconfiguration

Dynamic reconfiguration has been extensively investigated since the mid 80's. The research of J. Kramer and J. Magee ([83], [84]) has been among the first ones in the field of dynamic change in distributed computing systems. The approach of Moazami-Goudarzi ([85], [86]) extends the work of Kramer-Magee by dynamically building the component set involved in a reconfiguration. The approach of Bidan [87] addresses dynamic reconfiguration within the CORBA framework and supports multi-threaded objects. In that approach, the authors propose a new algorithm to perform dynamic reconfiguration that will minimize a distributed system's disruption time. To achieve this, their system maintains a directed graph of all connected objects. Gründer and Geihs's approach in [88] focuses on inheritance and reuse of object implementations and is based on the decoupling of object state and behavior. Almeida’s ([89], [90]) dynamic reconfiguration approach is also built on top of the CORBA framework but is even able to handle re-entrant requests. Senivongse's [91] approach uses mapping objects to adapt incoming requests from client objects to the current server object version. Other approaches rely on fault-tolerance characteristics to achieve dynamic reconfiguration. The approach of Tewksbury et al. ([92], [93]), also known as the Eternal System, is a CORBA based fault tolerant system using component replicas to perform reconfiguration. In Ajmani's approach ([94], [95]), the model assumes that long-lived systems are robust and that they tolerate node failures with correct object state recovery. Therefore, the model considers an upgrade as a soft restart. In this approach, a node can handle calls even if they have been made when the system was in a different version. To that effect, a node uses some request transformation functions to handle requests from a past version of the system or from a future version of the system, in case the calling node has upgraded before the receiving node.
3.2 Kramer and Magee

The research of J. Kramer and J. Magee [84] has been among the first ones in the field of dynamic change in distributed computing systems. They define the union of all elements of their dynamic configuration framework as a system. A system is composed of processing entities, the nodes, which may have directed communication paths, the connections, between them. As shown in Figure 29, a connection originates from an *initiator* and targets a *recipient*.

![Directed connection](image)

*Figure 29. Directed connection*

As shown in Figure 30, the approach separates two concerns:

- Functional application concerns which relate to the changes in functionalities of a software component independently of structural changes.
- Structural configuration concerns which relate to the structural changes independently of application state.

![Functional and structural concerns](image)

*Figure 30. Structural and functional concerns*
The authors point out that change is well handled at the configuration level with the use of software components (nodes) and their connections. They define change transactions as an ordered set of structural change operations. These change transactions are derived from change specifications.

As shown in Figure 31, the system evolves incrementally by the execution of the change specifications. Configuration management is the interface between the functional view of application programming and the structural view of system configuration. Changes are specified declaratively in terms of system structure only. The configuration management process uses the change specifications to define the new configuration of the system.

![Figure 31. Configuration management](image)

The role of change management is defined as the controlling of application consistency through changes.

Dynamic change management should comply with the following rules:

- Changes should be specified in terms of the system structure.
- Change specifications should be independent of the algorithms, protocols and states of the application.
- Changes should leave the system in a consistent functional state.
- Changes should minimize disruption of the application system.
The parts involved in the reconfiguration process must be able to reach a consistent state before changes are applied. This consistent state requires that there is no communication or transaction in progress between the affected nodes or between the nodes and their environment. Each node is then said to be in a quiescent state that must last through all the changes. A transaction is defined as an exchange of information between two and only two nodes and initiated by one of the two nodes. A transaction can comprise several message exchanges between nodes. A node can change the state of another node through a transaction. All transactions must complete in finite time. The model constrains all transactions to be independent. An independent transaction does not depend on any other transaction, such as nested transactions. Change is defined as a modification to the system structure. The possible change primitives comprise node creation and deletion, and connection addition and removal.

The model points out that management should be independent of any particular application, since the state of the application is represented by the structure of the nodes and the data. This is achieved through having each node abstract its state into a set of configuration management states. These configuration management states allow the management system to perform changes while leaving the application in a consistent state. The possible states for an application node are illustrated in the state transition diagram in Figure 32.

![Figure 32. Node state transitions](image)

There are two states for an application node:

- The active state, in which the node can initiate, accept and service transactions
- The passive state, in which the node can accept and service transactions, but not initiate any new ones. Furthermore, the node can’t be the initiator of any pending transactions.
Reconfiguration occurs during the passive state. However, the passive state is not sufficient for reconfiguration as the node may still be processing transactions initiated by other nodes. Since a passive node continues to serve incoming transactions, another state must be added for the consistency constraint. When a node is in passive state and when it is not or will not be part of any transaction, as recipient or initiator, it is said to be in the quiescent state. It is possible to reach the passive state and the quiescent state in finite time.

The possible change primitives are subject to constraints. The node deletion precondition is that it must be isolated. An isolated node is defined as a node that has no links to other nodes, whether it is as initiator or recipient. The precondition for link establishment or removal is that the initiator of the link must be in the quiescent state. The node creation precondition is that the node is isolated and therefore is in the quiescent state.

It is also possible to apply multiple reconfigurations simultaneously.

Limitations

The limitations of the model are the following:

- The application programmer must write all node implementations for the operations leading to the passive state.
- Small reconfigurations involving a few nodes result in substantial disruptions to the system.
- Re-establishment of application-state invariants\(^{21}\) is done through routines embedded in nodes.

\(^{21}\) As defined in [89]: "Application-state invariants are predicates involving the state (of a subset) of the entities in a system. The preservation of safety and liveness properties of a system depends on the satisfaction of these invariants. For example, let us consider a component that generates unique identifiers. An application-state invariant could be “all identifiers generated by the component are unique within the lifetime of the system”. In order to preserve this invariant, the new version of the component must be initialized in a state that prevents it from generating identifiers that have been already used by the original version. So, either (i) the set of all used identifiers is provided to the new version of the component, or (ii) the last used identifier is provided to the new version of the component. The latter alternative would require knowledge of the assignment mechanism used by the original version."
3.3 Moazami-Goudarzi
The approach of Moazami-Goudarzi ([85],[86]) extends the Kramer-Magee's one. This framework defines a distributed application as a finite number of processes executing on different hosts. All communication between these nodes is done through controlled directed communication paths. A process has a self-contained state except when it is part of a transaction. A transaction is defined as an interaction between two processes.

The configuration operations are create, link, unlink, and remove. The operations create and remove, add and destroy a process participating in the system, respectively. The operations link and unlink, establish and remove communication paths between the processes, respectively.

Before performing any of these operations on a node, the node must reach the safe state, which is when the node is not involved in any transaction. The safe state differs depending on the operation:

- For the create operation, there are no restrictions, it is always the safe state.
- For the link and unlink operations, the safe state is reached when the starting point of the communication path is not part of a transaction.
- For the remove operation, the safe state is reached when the node to be removed is isolated. A node is said to be isolated when there are no communication paths starting from or arriving at it.

The author limits his model to a deadlock-free, non-cycling, non-interleaved (transactions) system. With these constraints, he states that each transaction executes as if it were the only ongoing transaction in the system. Transactions affecting non-overlapping nodes can run in parallel. Multiple transactions on the same node run sequentially.

The approach presents 2 algorithms to drive the system in the safe state:

- The first one imposes the safe state to all nodes of the system.
- The second one uses a starting set of nodes to be blocked, the BSet, and adds dynamically other nodes that are currently part of a transaction or sub-transaction from this set of starting nodes.

3.3.1 Algorithm 1: Safe state for all nodes in the system
The author shows that it is always possible to reach the safe state for such a system. As shown in Figure 33, to reach the safe state, a software entity called the coordinator node sends a block message to all the nodes in the system. When all
nodes respond with an acknowledgment block message, the system is in the safe state. When receiving the block message, a node that is in the idle state goes immediately in the blocked state. If a node is in the active state, thus currently involved in a transaction, the node will complete the transaction before returning to the idle state and then to the blocked state. No new transactions can start from nodes in the blocked state. In case a node in the blocked state receives a transaction request, it moves into a so-called unblocked state. In the unblocked state a node can continue the execution until fulfillment of the unblocking request. Once an unblocked node has finished executing the request, it changes again to the blocked state.

![Figure 33. Modified state transition diagram](image)

The author also points out that since every transaction executes as if it were the only one in the system, and since it is possible for each transaction to reach the safe state, this will also work for several non-overlapping simultaneous transactions.

3.3.2 Algorithm 2: Blocking Set (BSet) of nodes

In this case only the nodes that need to get reconfigured shall get blocked. At the beginning of the algorithm, the BSet contains only the nodes to be reconfigured as shown in Figure 34. This is called the original BSet. To reach the safe state, the first strategy to be thought about is to block all requests targeting BSet nodes from non-BSet nodes. But this can lead to a deadlock in case there is a BSet node that must first complete its transaction and has sub-transactions going through non-BSet nodes which eventually have also sub-transactions that involve BSet nodes. In this case there would be a pending transaction that can not be completed.
A BSet node has an ongoing transaction with a non BSet node (the request was initiated before reconfiguration started)

Figure 34. Original BSet

To avoid such a deadlock situation but still blocking only a subset of nodes during reconfiguration, the BSet is expanded with the calling flow. As shown in Figure 35, all nodes that are targets in transactions where the source is a BSet node become themselves part of the BSet. This is called the extended BSet.

The node gets added to the BSet and initiates a request to another node (subtransaction)

Figure 35. Extended BSet with nested sub-transaction

A subsequent sub-transaction from this newly added BSet node will also add the sub-transaction target to the BSet as shown in Figure 36. This will continue until there are no more new nested sub-transactions.
The node gets added to the BSet. The transaction and subtransaction will complete before the system will be in safe state.

Figure 36. Extended BSet with no further sub-transaction

The other rule is that all incoming requests to BSet nodes that originate from non-BSet nodes are blocked. This permits to reduce the problem to the first case and can then get solved with algorithm 1 considering that the dynamically enlarged BSet is an entire system in the safe state since there are no pending transactions involving any of the BSet nodes.

In case there are some simultaneous requests originating from inside and outside the BSet at the start of the configuration, the requests from outside the BSet are first blocked as shown in Figure 37.

If a node out of the original BSet becomes part of the extended BSet because it becomes the target of some transaction originating from a node in the BSet, it becomes also part of the BSet as shown in Figure 38.
This node is added to the BSet because it has an incoming request of a BSet node.

**Figure 38. Extended BSet with blocked request**

Once this node has become part of the BSet, the blocked request can resume as shown in Figure 39.

**Figure 39. Unblocked request**

When all the nodes in the original BSet are in the blocked state, this implies that all the transactions or sub-transactions which have as all-first initiator a node from the original BSet have been completed. This means that the system is in the safe state when considering only the nodes involved in the reconfiguration and this even if there were some other transactions pending in other nodes in the extended BSet which have an initiator that is not part of the original BSet. Therefore the reconfiguration can occur immediately as shown in Figure 40. It is not necessary to wait until the BSet dynamically added nodes also acquire the blocked state. The BSet dynamically added nodes can get unblocked as soon as all the original BSet nodes are blocked.
As soon as the "O" transactions complete the system is in safe state for reconfiguration, even if there are still "X" transactions in progress.

Figure 40. Safe state is reached when all original BSet nodes are blocked

Limitations

Interleaved transactions are not permitted but the authors suggest that the non-interleaving style of programming makes debugging and maintenance simpler and allows the operating system to make more optimal decisions regarding the use of resources.

The type of distributed systems this model can be applied to is very limited, since each node can treat only one transaction simultaneously. This leads to the exclusion of re-entrant and multi-threaded systems.

3.4 Bidan et al.

The approach of Bidan et al. [87] addresses dynamic reconfiguration in the CORBA framework and supports multi-threaded objects. The authors propose a new algorithm to perform dynamic reconfiguration that will minimize a distributed system's disruption time. They extend the CORBA LifeCycle Common Object Services facilities. They define dynamic reconfiguration as the changes of the system’s logical and physical structure which is comprised of insertion, removal or replacement of physical nodes, logical components and interaction links. The authors classify these changes into two categories:
• Programmed reconfiguration that is part of the system design and is scheduled to happen when certain conditions are satisfied during application execution (i.e. embedded failure recovery)
• Evolutionary reconfiguration that refers to structure modifications caused by events independent from the application specifications like components upgrade, etc.

They argue that both situations can be handled by the same primitive operations. These operations are the following:

• Creation and removal of components.
• Creation and removal of links.
• State transfer among components.

The model proposes the use of a so-called Dynamic Reconfiguration Manager (DRM). The DRM maintains a directed graph of connected objects. An object A is linked to an object B if A can invoke a method on B. The DRM will block all reconfiguration-involved links during a reconfiguration activity. The DRM initiates and coordinates all reconfiguration activities of application objects. The authors define local node consistency as RPC\textsuperscript{22} integrity in the CORBA framework. RPC integrity is itself defined as the completion of all calls between CORBA objects which are part of a reconfiguration activity before the reconfiguration changes are made. They state that their model guarantees local node consistency but not application consistency, which would comprise the atomic reconfiguration of a set of objects. They propose that additional mechanisms should be built on top of their reconfiguration service to guarantee application consistency. Their approach does not use reconfiguration points since they argue that this method is too error-prone in distributed systems. They have oriented their research to define an algorithm that is application independent and that can guarantee local consistency while minimizing execution disruption time. Although their algorithm is designed only for object configurations with no non-reentrant or non-cycle calls, the authors describe very briefly a reentrant algorithm but they state that a "big majority of applications is based on cycle-free object interactions".

To achieve their goals, they define the primitives create, remove, link, unlink, transferLink, and transferState. These primitives are used to respectively create and destroy a link, transfer the requests pending on a passivated link to another existing link, and to transfer the state from one object to another. As shown in Figure 41, these primitives are the building blocks for their reconfiguration algorithm.

\textsuperscript{22} As described in section 2.6.2.
Reconfiguration =

\begin{verbatim}
var
  % data object that stores the configuration graph
  config: config desc;
begin case
  create(obj: obj desc):
    config.addObj(obj); create(obj);
  link(client obj: obj desc; server obj: obj desc):
    config.addLink(client obj, server obj);
  unlink(client obj: obj desc; server obj: obj desc):
    passivateLink(client obj, server obj);
    config.delLink(client obj, server obj);
  transferLink(client obj: obj desc; server obj1, server obj2: obj desc):
    if (passive(client obj, server obj1)) then
      moveLink(client obj, server obj1, server obj2);
    else
      moveLink(client obj, server obj1, server obj2);
  remove(obj: obj desc):
    blockObject(obj); config.delObj(obj); remove(obj);
  transferState(obj1, obj2: obj desc):
    blockObject(obj1); copy(obj2, getState(obj1));
    unblockObject(obj1);
end case
end
\end{verbatim}

\textbf{Figure 41.} Reconfiguration algorithm for Bidan’s approach

On the implementation side, all reconfigurable objects must inherit from a particular class called \textit{RO\_object}. This is a prerequisite for an object to be part of the reconfiguration process. This class handles all the low-level work for an object to be reconfigurable. Moreover, each reconfigurable object redefines the ORB \textit{invoke} operation that handles RPC requests to other objects. This operation is redefined to issue the RPC request only if the link is valid and not passivated.

The authors also analyze the performance of a reconfigurable system relative to a non-reconfigurable system. They set up 5 object systems with a communication intensity ranging from 5\% to 95\% of the total execution time. As shown in Figure 42, the results show a difference performance from 1\% to 45\%.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
\textbf{Communication intensity} & \textbf{Non-Reconfigurable} & \textbf{Reconfigurable} \\
\hline
95\% & 18.31 sec & 26.59 sec \\
50\% & 32.00 sec & 38.54 sec \\
35\% & 47.27 sec & 54.12 sec \\
20\% & 79.07 sec & 80.11 sec \\
5\% & 408.08 sec & 408.12 sec \\
\hline
\end{tabular}
\caption{Performance results}
\end{table}
3.5 H. Gründer and K. Geihs
The approach proposed by H. Gründer and K. Geihs in [88] focuses on inheritance and reuse of object implementations. The approach is based on the decoupling of object state and behavior. The object state and identity is implemented in an object called "chassis". The different object behaviors are implemented in separate objects called "engines". An object can therefore have several implementations simultaneously as shown in Figure 43. This separation of the object into two different objects is only relevant from an engineering viewpoint. It is transparent to the clients using remotely the conceptual object.

When a new service interface must be built, it is possible to aggregate several existing service interfaces and add some new methods and attributes to the newly specified service interface. It is also possible to reuse the engines of the existing interfaces or to specify a new behavior for one of the interfaces while keeping the other interfaces with their original engines. The interface definitions are described with an IDL. In this case the software evolution is provided by creating new interfaces and rewriting the corresponding implementations. The system is more aimed at offline evolution than dynamic reconfiguration even if the authors claim that several object engines could run simultaneously for a single object. This could lead to a possible dynamic software evolution system.

23 Interface Definition Language
3.6 Twittie Senivongse

The approach of Twittie Senivongse outlined in [91] is based on the Reference Model for Open Distributed Processing (RM-ODP)\textsuperscript{24}. The RM-ODP is a joint effort by the International Standards Organization and the International Telecommunications Union (ITU) to provide a coordinating framework for the standardization of Open Distributed Processing (ODP) by creating an architecture which supports distribution, interworking, interoperability and portability. This extension of the RM-ODP adds evolution transparency to RM-ODP distribution transparencies [96]. It points out that the most used distributed platforms like CORBA, DCE or RM-ODP only authorize transparent subtype evolution. The model is based on a basic client request system as shown in Figure 44 and extends it by adding versioning capabilities.

![Figure 44. Mapping functions](image)

The proposed model favors a dynamic request upgrade approach as shown in Figure 45 instead of a dynamic client upgrade solution as shown in Figure 46. The dynamic request upgrade approach provides evolution capabilities without changing the clients that use previous service object interface versions. This approach permits to evolve a service interface not only on a type-subtype basis, but also when the new interface version is syntactically incompatible to the old versions.

![Figure 45. Dynamic request upgrade](image)  
![Figure 46. Dynamic client upgrade](image)

\textsuperscript{24} Please refer to Appendix B in the appendices for more information on RM-ODP.
To achieve this, the model uses *mapping operators* to chain the invocations of clients on old server types to new server types. These *mapping operators* are objects that impersonate the old services and act as mediators between the clients and the new version of the service. The mapping operator object acts as an indirect binding object that sets up the communication path between the client and the server as shown in Figure 47. The mapping operator does not manage quality of service.

![Figure 47. A mapping operator acts as a binding object](image)

The correspondence between the versions of the same service type is achieved by the use of *mapping functions* which represent can-be-mapped relationships. *Mapping functions* identify how operations, parameters, and values in a version of the interface relate to those in another version as shown in Figure 48.

![Figure 48. Mapping functions](image)

The complete model makes usage of a *trader* element which is used to obtain the correct *mapping operator* for a call to interface A. The mapping operator can then itself obtain a reference to the new interface A' of the server. The server can export to the *trader* a new version of the interface and retrieve the previous one as shown in Figure 49.
Managing evolution risk of cross-organizational services

The model does not require that all service objects of a same type evolve at the same time. The versioning is therefore not at type level but at instance level. A version is defined as a stable and coherent data and implementation state which the service provider desires to keep. An interface version is never itself updated. Rather than changing a version, a new version is created that contains the desired changes. An existing version is left untouched. The model assumes that an original service instance will cease to exist after its type has evolved and a new service instance has replaced it. A service instance does not exist in several versions simultaneously. The model does not permit to handle simultaneously two different versions of a type that have the exact same interface. The author suggests that a possible solution could be the attachment of version numbers or timestamps to versions of the interface definitions.

The model also assigns to the service interface evolver the responsibility of converting the service object instances. A service interface evolver is meant to be a person. The author points out that a service interface evolver does not necessarily have all the authorizations to evolve all instances of an object type in a distributed environment.

The prototype implementation provides an evolution manager which is a software tool that facilitates and automates the process that responds to service evolution. The evolution manager controls the generation of mapping functions and mapping operators with help from a version repository called history database. As shown in Figure 50, the mapping operators and functions (or function skeletons in case of syntactic incompatibilities) are generated from the comparison between the interface versions. In case of syntactic and semantic incompatibilities, it is the service interface evolver that provides code for the mapping function.
3.7 Ajmani

Ajmani’s approach ([94], [95]) states that an automatic upgrade system must:

- Provide service while the nodes that comprise the system run different versions.
- Propagate the upgrades to nodes automatically.
- Provide a way of controlling the node upgrades.
- Provide a way to preserve the persistent state of nodes through versions.

The model assumes that long-lived systems are robust and that they tolerate node failures with correct object state recovery. Therefore, the model considers an upgrade as a soft restart. The model is composed of an upgrade infrastructure, scheduling functions, simulation objects and transform functions. The upgrade infrastructure is responsible for the upgrading process control and monitoring. The scheduling functions are implementations of upgrade policies that indicate when a node shall be upgraded. Simulation objects are adapters that simulate the behavior of other versions of an object. Transform functions are responsible for converting the state of an object from one version to another.
The model states that there is one object per node and that it has an identity and a state. An object is self encapsulated. All inter-object communication is done through method invocations. Each object can recover its persistent state from failure. Each object is derived from a top-level class. A version is defined for the whole system, not for an individual node. An upgrade is defined as the move from one version to another. The model assumes that upgrades are rare, such as once a month. The model also considers that the new object versions are fully debugged. All objects of a type involved in an upgrade must switch to the new version.

The system possesses a centralized upgrade server that maintains a version number for the whole system as shown in Figure 51. For each upgrade, this version number is incremented. When a node makes a call to another node, an upgrade layer labels the call with the version number. In each node, the upgrade layer makes queries to the upgrade server and reads also the version number of the incoming calls. If a node receives an incoming call with a higher version number, the node’s upgrade layer asks its local upgrade manager to download the upgrade information set from the upgrade server. This upgrade information set contains the following elements: old-class, new-class, TF, SF, past-SO, future-SO. Old-class identifies the class to be upgraded. New-class identifies the class that replaces old-class. TF is a transform function that transforms the persistent state of the object of the old class in the new class. SF is a scheduling function to parameter when the effective upgrade shall take place. Past-SO is a class for the simulation object that provides backward compatibility to the currently installed class. If a node receives from another node a call with a labeled version preceding the receiving node's version, then the past-SO of the receiving node will handle the call. Similarly, future-SO is a class for the simulation object that provides forward compatibility for the currently installed class. If a node receives from another node a call with a labeled version subsequent to the receiving node's version, then the future-SO will handle the call. The future-SO can be downloaded on-the-fly and therefore the node can handle calls of newer versions before it has been really upgraded.
After the upgrade download, the node's upgrade manager checks if the upgrade concerns the node. In case the node is not affected by the upgrade, it simply advances the system version number present in the node. In case the node is affected, the upgrade takes place in the node.

If the upgrade concerns the node, the upgrade manager launches in an independent processing thread the scheduling function. The scheduling function can access a centralized upgrade database to coordinate its upgrade with other nodes. At the determined upgrade time, the scheduling function informs the upgrade manager that the upgrade must take place. The upgrade manager shuts down the node, forcing its state to be persisted. The upgrade manager then installs the new class version. The new class version is not necessarily a subtype of the older class version. The upgrade manager uses then the transform function to convert the object’s persisted state to the representation of the persistent state of the new class. This is necessary since the persistent state representation can change between the versions of an object. The upgrade manager then installs the past-SO object that will be used for incoming calls that are labeled with an old version number. This is necessary since the upgrade can take place during run-time. The upgrade manager also removes the future-SO since the new class version will handle these incoming calls. The upgrade manager then restarts the node. An object of the new class is instantiated and reads its transformed persistent state.

The scheduling functions are procedures running in each node. They can implement different scheduling scenarios. Examples of these scenarios include immediate upgrade, one-replica-at-a-time upgrade, gradual upgrade, class c1 upgrade before class c2 upgrade, etc. It is also possible to coordinate centralized upgrades with the upgrade database since it contains all node version information.

Simulation objects are wrapper objects. They provide the required interface of an...
object for a specific version. Simulation objects delegate most of the processing to either another simulation object, or to actual object instances. In case there are several version changes in a short time, there can be a chain of simulation objects for future versions calls and for past versions calls as shown in Figure 52.

![Figure 52. Chain of future and chain of past simulation objects in a node](image)

Simulation objects can contain some state that is specific to a class version. They must therefore be able to persist their state and recover it in case of failure. Simulation objects must also initialize their state when they are installed. Future-SOs are initialized without any input. They may have some specific version state that can be accessed by the transform function when the persistent state of an object is transformed into the new state representation for the new object version as shown in Figure 53.

![Figure 53. The transform function can use a future-SO’s state to create the new persistent state](image)
It can happen that there are incompatible changes between version objects that simply do not allow any wrapping. In that case the calls will fail but since the model assumes the system is robust, i.e. it can recover from failures, the responsibility of applying such type of changes and the definition of their corresponding schedule must be left to the person in charge. Simulation objects are dynamically installed and discarded. They are dynamically installed when an incoming call has a newer version label than the current node’s version. They are discarded when the node’s upgrade manager observes in the upgrade database that the particular version is not needed anymore.

### 3.8 Eternal System, Tewksbury et al.

The approach of L.A. Tewksbury is based on the Eternal System ([92], [93]). The Eternal System is a CORBA-based fault tolerant system using object replicas. The Eternal System replicates objects, distributes the object replicas across the system (usually multiple hosts) and maintains consistency of the object replicas. The replicas of an object form an object group. A client object can invoke methods on a server object group as if there would exist a single server object. Client objects can also have replicas. The authors argue that the developer can create applications that benefit from object replication transparently.

As shown in Figure 54, the Eternal System is comprised of four different distinct elements: the Interceptor, the Replication Manager, the Resource Manager and the Evolution Manager. The CORBA Object Request Broker packages the requests according to the Internet Inter-Orb Protocol (IIOP). These IIOP requests are then captured by the Interceptor before they are transmitted through TCP/IP. The Interceptor is positioned between the ORB and the operating system network interfaces. The IIOP messages are intercepted after they leave the Object request Broker but before they reach the Operating System TCP/IP library [97]. They are then rerouted to the Replication Manager, which passes the messages to a group communication system that multicasts the messages to the object replicas. The Replication Manager is responsible for dispatching transparently the invocations done on an object group to all object replicas. To achieve this, it uses a so-called group communication system that provides a reliable, totally ordered message delivery system. The Replication Manager also maintains the consistency of the replicas of an object. The Resource Manager allocates the replicas to the different hosts to satisfy the fault tolerance and load balancing objectives. The Evolution Manager is responsible for performing live upgrades of objects in an atomic fashion. The evolution of objects takes place while the system continues to run. The Resource Manager and the Evolution Manager are CORBA objects. The Interceptor and the Replication Manager are non-CORBA pieces of software.
The online evolution of objects is possible due to the object replication capabilities of the Eternal System. There are two types of object replication schemes in the Eternal System, the passive replication and the active replication.

In active replication, an invocation from a client object to an object server group is dispatched to all the server object replicas as shown in Figure 55. All server object replicas execute the invocation and return the response. The Replication Manager sends back to the client only one response since they are all identical. In active replication, if there is a failure, removal or upgrade of an object replica during a method invocation, the system continues running because the other replicas also execute the method. The result from another replica will simply be returned to the client object.

Figure 54. Overview of the Eternal System
In passive replication, when a client object invokes a method on an object server group, the invocation is multicasted to all replicas. All replicas write the method invocation with all parameter values to an invocation log. However, only one replica, the primary replica, will really process the method invocation as shown in Figure 56. In case of a failure, removal or upgrade of the primary replica, another replica will be promoted as the primary replica and will then execute the method invocation using the information stored in the invocation log. In case of failure, removal or upgrade of a replica other than the primary replica, the system simply continues its execution. In all cases, the Replication Manager detects and suppresses duplications of method calls and responses. To achieve this, it labels each call or response with a unique identifier.

In case of normal execution, the passive replicas will have their state updated by the Replication Manager with the state of the active replica that just successfully executed the method invocation. This state transfer from the primary replicas to the other replicas is transparent to the client. The state transfer can be either automated for the simplest cases or the developer must write some state transfer code. State transfer must take place when the object is quiescent, i.e. when there are no active threads within it.
The Evolution Manager uses the replication capabilities to perform upgrade or evolution of objects. As shown in Figure 57, the Evolution Manager is comprised of two parts, the Preparer and the Upgrader. The Preparer is a tool that helps create an intermediary object class from an initial object class and a target object class. The intermediate object class will contain usually the data and the methods of both initial and target object classes. The Preparer analyzes both initial and target classes and proposes the developer an intermediate class. Some cases can be handled automatically, but the most complex ones are left to the developer who must then provide the state transition code. In some cases, it can be necessary to have several subsequent intermediate versions, because the difference between the initial and target object classes is too big to be bridged in a single step.

The Upgrader is responsible of replacing the objects by their new versions. The Upgrader first replaces all designated initial objects by objects of the intermediate class. During this replacement, there is always at least one available object offering the initial methods, i.e. there is either an object of the initial object class or there is an object of the intermediate object class (which offers both the initial and the target methods). When replacing an object of an initial class by an object of the intermediate class, there is a state transfer from the initial class object to the initial part of the intermediate object. Once all objects of the initial object class have been replaced by objects of the intermediate object class, there is an atomic switchover to the new version of functionalities. From this point on, the intermediate objects behave as if they were objects of the target object class. During this atomic switchover, there is a state transfer within the intermediate objects so that the state of the initial part of the intermediate objects is coherent with the expected state of the target part of the intermediate objects. The Upgrader can then replace the objects of the intermediate object class with objects of the
target object class. During this operation, there is again always at least one object of the intermediate class or of the target class that is available to fulfill incoming requests. Again, when replacing an intermediate object with an object of the target class, there is a state transfer between the target part of an intermediate object and the new object of the target class. All these replacements can be done on an object per object basis, without disrupting the global application execution. The only point at which the application is disrupted is during the atomic switchover, where all involved objects must be in a quiescent state.

There are two major cases that should be considered for an object class upgrade or evolution: the interface preserving evolution and the interface changing evolution.

The interface preserving evolution is the simplest of the two cases. A client object group makes method invocations to a server object group. The server object group must be replaced with a new class version. The new class version has the same interface as the current class version. As shown in Figure 58, there are four different object classes involved in the upgrade: the client object class, the initial server object class, the intermediate server object class and the target server object class.
The Upgrader performs then the steps to migrate the server objects from the initial version to the target version. All the object replacements are done while the individual replaced objects are quiescent. As shown in Figure 59, there are several processing phases during this process:

- **Phase 1.** The client object group invokes a method m on the server object group that contains only objects of the initial server object class.
- **Phase 2.** Each server object is replaced one at a time by a server object of the server intermediary class. During this operation, the intermediate server objects behave like server objects of the server initial class. A client object invocation of the method m of the intermediary server object executes the "initial version" code of the method. During this phase there are server objects of the initial class and the intermediate class in the same object group.
- **Phase 3.** In this phase all initial server objects have been replaced by intermediate objects. The system continues to run.
- **Atomic switchover.** When all server objects of the initial server class have been replaced by server objects of the intermediate server class, the atomic switchover takes place in all server objects. During this atomic switchover, there is a state transfer between the "initial version" state part and the "target version" state part of each server intermediate object.
- **Phase 4.** From this point on, the intermediate server objects behave like server objects of the target server class. The client objects still invoke the method m on the server objects but now it is the code of the "target version" part of the intermediate object that is executed when m is called.
- **Phase 5.** The intermediate server objects are replaced one at a time by server objects of the target server class. During this phase there are both intermediate objects and target objects in the server object group.
- **Phase 6.** The upgrade is completed once all intermediate server objects are replaced by server objects of the target server class.
Although the interface changing evolution is a more complex case, it can be treated in a similar fashion. A client object group makes method invocations to a server object group. The server object group must be replaced with a new class version. The target server class version has a different interface than the initial server class version. As shown in Figure 60, there are six different object classes involved in the upgrade: the initial client object class, the intermediate client object class, the target client object class, the initial server object class, the intermediate server object class and the target server object class.

Figure 60. Classes involved in interface changing evolution

In this case, the Upgrader then performs the steps to migrate the client and the server objects from the initial versions to the target versions. Again, all the object replacements are done while the individual replaced objects are quiescent. As shown in Figure 61, there are several processing states during the replacement phase:
- **Phases 1 to 3.** These three phases are exactly the same as the first three phases of the interface preserving evolution case.

- **Phase 4.** Each initial client object is replaced one at a time by an object of the intermediate client object class. There is always at least one client object that is running (not quiescent). The running client object can be either an initial or an intermediate client object. The intermediate client objects behave like initial client objects and execute therefore the “initial version” part of their code. During this phase there are initial and intermediate client objects in a client object group.

- **Phase 5.** All initial client objects have been replaced by intermediate client objects which run the “initial version” part of their code. The system continues normally its execution.

- **Atomic switchover.** When all server objects of the initial server class and all client objects of the initial client class have been replaced by their intermediate object counterparts, the atomic switchover takes place in all intermediate client and server objects. During this atomic switchover, there is a state transfer between the "initial version" state part and the "target version" state part of each intermediate client and server object.

- **Phase 6.** Each intermediate client and server object executes the “target part” of its code. A target client object will therefore call a server object’s m’ method instead of its m method. The server object runs then the new m’ method.

- **Phase 7.** Each intermediate client object is replaced by a target client object on a one by one basis. There is always at least one intermediate or target client object running. During this phase, there are intermediate and target client objects in a client object group. All server objects are intermediate server objects. The system continues running without interruption.

- **Phase 8.** All intermediate client objects have been replaced by target client objects. All server objects are intermediate server objects. The system continues running without interruption.

- **Phase 9.** Each intermediate server object is replaced by a target server object on a one by one basis. There is always at least one intermediate or target server object running. During this phase, there are intermediate and target server objects in a server object group. All client objects are target client objects. The system continues running without interruption.

- **Phase 10.** All intermediate server objects have been replaced by target server objects. All client objects are target client objects. The system continues running without interruption. The reconfiguration process is complete.
Managing evolution risk of cross-organizational services

Limitations

The use of the Interceptor which is an operating system interceptor, yields mixed outcomes. One main advantage is that it is ORB independent which permits to use any commercial ORB. A disadvantage is, that since the Interceptor catches only the IIOP messages before they reach the OS TCP/IP libraries, it can't intercept invocations between co-located objects [97]. To insure strong replica consistency, the Eternal system enforces that an object services only one invocation at a time. The system also excludes re-entrant invocations. The Atomic switchover feature also forces the initial, intermediate and target object classes to be written in the same programming language.

3.9 Almeida

The dynamic reconfiguration approach of Almeida ([89], [90]) is built on top of the CORBA framework. It is characterized by the so-called Dynamic Reconfiguration Service (DRS). In this approach, a system configuration is defined as a structure of objects. As shown in Figure 62, Figure 63, Figure 64 and Figure 65, the dynamic reconfiguration process can be comprised of the following operations on these software entities:

- Addition of an object.
- Removal of an object.
- Replacement of an object by another.
- Migration of an object to another location.
Figure 62. Addition of an object

Figure 63. Replacement of an object

Figure 64. Removal of an object

Figure 65. Relocation of an object
The design steps for the reconfiguration process are:

- specification of the changes in terms of entities and operations
- specification of constraints restricting the reconfiguration process

The approach defines the notion of a “correct” state for a system. This “correct” state is defined by the fulfillment of 3 conditions:

1. **The structural integrity** is preserved. *Structural integrity* is fulfilled when the clients obtain the object reference of the new object, and when the new version of the server object is a subtype of the old version (the interface definition is backwards compatible).
2. The entities are in mutually consistent states.
3. The application state invariants hold.

Entities are said to be in mutually consistent states when they are left in a defined predictable state after an interaction between them. Interactions are the only way an object can change the state of another object. The safe state is reached when all the entities involved in a reconfiguration are in a consistent state and when there are no ongoing interactions between them.

The author identifies possible reconfiguration approaches and classifies them using the tree shown in Figure 66.

![Figure 66. Classification tree for reconfiguration approaches](image-url)
The author proposes a mechanism that assures that ongoing interactions complete before reconfiguration takes place. This is achieved by driving the system to a reconfiguration-safe state.

The proposed algorithm uses the middleware platform information at runtime and freezes system interactions on-demand. There are three defined steps in the algorithm:

1. The system completes its interactions with the objects involved in the reconfiguration and reaches the safe state. All new interactions that would use these objects are blocked.
2. The system gets confirmation that the safe state has been reached.
3. The reconfiguration process is applied.

The reconfiguration process takes place through so-called reconfiguration steps. Two types of reconfiguration steps are defined:

1. The simple reconfiguration step which is one reconfiguration operation on one object.
2. The composite reconfiguration step that comprises several reconfiguration operations on several objects as shown in Figure 67.

For a simple reconfiguration step, all requests issued to the object are queued by the middleware platform. When all ongoing requests have been completed, the system is in the safe state. A re-entrant request is not queued, because the object would have a pending outgoing request that would never be completed as shown in Figure 68.
Managing evolution risk of cross-organizational services

Figure 68. Simple reconfiguration step. Requests that should not be queued.

For a composite reconfiguration step, several objects have to be driven to the safe-state. As shown in Figure 69, requests issued by an affected object or, as shown in Figure 70, nested requests that are a consequence of requests issued by an affected object are never queued because there would always be an incomplete pending outgoing request in the set of affected objects.

Figure 69. Unqueued requests issued by an affected object
The middleware platform is responsible for selectively queuing requests that belong to the blocking set and for allowing requests that belong to the ‘laissez-passer’ set. For each reconfigurable object in an invocation path, the middleware infrastructure adds the object’s identification to the request as an implicit parameter. This is done transparently for the application objects. If at least one of the affected objects has been included in the request’s implicit parameters, the request belongs to the ‘laissez-passer’ set, and should be allowed to complete. After reconfiguration, queued requests and new requests are redirected or use the new or relocated version of an object. The system provides a central point of contact to find the new or relocated object called the Location Agent. The Location Agent will map the new objects to the invalid object references.

Limitations

The limitations of the approach can be summarized as follows:

- All the invocations pass through a centralized node, the DRS.
4 Cross-organizational interactions

An organization is defined as “a social unit of people, systematically arranged and managed to meet a need or to pursue collective goals on a continuing basis. All organizations have a management structure that determines relationships between functions and positions, and subdivides and delegates roles, responsibilities, and authority to carry out defined tasks. Organizations are open systems in that they affect and are affected by the environment beyond their boundaries.”[98]

Thus, each organization pursues its own goals, whether it is a profit-oriented company, a non-governmental organization or an administration. But all these organizations tend to be more and more interconnected [99]. Their business processes are increasingly integrated and therefore always more interdependent. The integration of these processes between interacting organizations implies that the exchange of information relative to these processes is also integrated. Data are exchanged between these organizations, such that each of them can achieve their own goals.

As defined previously, organizations have a management structure. This management structure defines the roles and responsibilities of the individuals who are part of the organization. The management also exercises authority on how the organization is structured and how it functions. The management also determines what information is necessary to accomplish its goals and decides on the best ways of implementing these data flows. In this context, the choice of an organization's information system is left to the organization itself. It is a self-contained decision.

But, when organizations want to exchange data between each other, they face the basic problem of the authority's ownership. In other words, interacting organizations must decide which of them has the power of defining the nature of the data flows among them, the way these data flows will be exchanged and the timing of all related tasks and events of these data flows.

To answer these questions, organizations have often chosen between two distinctive approaches. This has been usually dependent on the degree of correlation of the goals pursued by the interacting organizations. Hence, organizations tend to interact either through an alliance approach or through an archipelago approach.

25 It delegates this task to its analysts, and decides eventually on an implementation based on the analysts' recommendations.
26 When will the data flows be functional, when do they end, etc.
4.1 Alliance approach

In case the goals of multiple organizations are similar, and if they still have to coordinate between each other to satisfy the needs of their respective customers, the number of interactions and connections between the different actors becomes then quickly very important as shown in Figure 71. If each organization has its own data exchange protocols, the effort to create and maintain automated data flows between the involved organizations becomes rapidly very expensive as their number grows. This is due to the exponential number of adaptations that must be performed by all the involved organizations as new organizations join the group. If the organizations exchange data mostly for cooperation and coordination to satisfy customer needs\(^{27}\), then the organizations could form *alliances*, which are often materialized by the creation of joint-ventures [100]. A joint-venture is effectively, the creation of a new organization, in which each alliance member has a governing voice [101]. The goals of this new organization is to centralize control, reduce complexity by creating and implementing standards and reduce operating costs ([100], [102]).

![Figure 71. Several organizations with very similar goals needing to coordinate and cooperate with each other without a centralized system](image)

Each participating organization of the alliance outsources to this newly created organization the data and the processes that are needed to be shared by all members of the alliance. The goal of the new organization is to execute the delegated tasks for the sole benefit of its owning members. As shown in Figure 72, the members of the alliance use the centralized system that has its own

\(^{27}\) The customers being outside of the alliance.
management but which is under control of the alliance members. The consolidation in some markets like the airline industry and the regrouping of many of its actors has led to the creation of some of the biggest alliances [100]. Since the alliance's members outsourced to the joint-venture essentially all of their common information processes, this has led to the creation of some of the largest distributed systems.

![Diagram](image)

**Figure 72. Centralized system of an alliance approach**

The most compelling examples of this approach are the Computer Reservation Systems (CRS) used to book train and flight tickets and soon-to-be-followed Global Distribution Systems (GDS) ([103], [104]). There are four leading GDS's today: Amadeus, Galileo, Sabre, and Worldspan. These GDS's are independent companies owned by their members which are airline and train companies, car rental firms, and hotel groups.

This approach of externalizing parts of a system that needs coordination between all involved actors is still being advocated in research [105]. The only major proposed change is that the centralized system can be interfaced through services. Nevertheless, it is still a centralized system under a single authority that offers remote interactions. This means that although this approach is decades old, it has proven to be very effective in some specific domains. Thus, it has been questioned if radical changes to new models or technologies in travel-related information systems are worth the risk [106].
4.2 Archipelago approach

In addition to the alliance approaches in which a dedicated joint-venture is created, interacting organizations still need to cooperate with each other. They have goals which eventually produce goods and services that are complementary, from the economy's²⁸ point of view. Thus, they must also try to maximize their productivity. This often implies the integration of their business processes and consequently, the automation of the information exchange necessary to that integration. These organizations are all independently managed and they have all made their own choices about their information technology stack. Their plannings are also completely independent. The life cycles of their respective applications are different. The individuals managing each of these companies are only accountable for what happens within the boundary of their own company. From the wider perspective, it is a system where there are no incentives for global optimization. Although it may be possible for two or three companies to agree on a common information technology stack so that they may be able to integrate easily, it would still not be applicable. This is because these companies do not work exclusively with each other but also with other organizations not part of that group. If it would be needed to use the same technology stack to work with each other, then these other partners would also need to agree to use the same technology stack. Thus, if using the same technology stack would be mandatory for organizations working with each other, then by extension all existing companies would need to agree eventually to use the common one, which is not possible because it would mean that there exists only a single technology stack among all organizations²⁹. Therefore, companies must find ways to integrate their data flows in some other way.

This has led to the situation that each company shields its information technology stack from the outside world by establishing more and more technology-agnostic data flows with its partners. This can be compared to an archipelago of software islands³⁰ ([109], [110], [107], [108]).

At the intra-organization level, the islands are individual applications that are communicating with each other through standardized or proprietary technologies and protocols. Since each application of the organization usually communicates only with a subset of the rest of the organization's applications, a graphical representation of the network of interconnected applications is similar to an archipelago as shown in Figure 73.

---

²⁸ At the highest level of abstraction.
²⁹ The fact is, this is not the case.
³⁰ A software island is defined in a general context as some remote piece of functionality accessed by clients [107]. In the Service Oriented Architecture context, it is more specifically defined as services being accessed by clients [108].
At the cross-organizational level, we define an island not as an individual application but as the *complete organization itself*. Thus, in this context we do not consider the dependencies between applications but between organizations\(^{31}\) as shown in Figure 74. At this level, each island is subject to *internal changes* that can not be perceived by the island's outside world, and to *external changes* that can be perceived by the island's outside world. Thus, internal changes that occur within each island do not affect directly the other islands. Only changes to what an organization offers outside its boundaries should affect its dependent siblings. Since each island has its own management, each island has its own pace of evolution.

---

\(^{31}\) Even if technically the dependencies are between applications.
Over time, the number of cross-organizational data flows has increased. Since the communication between each pair of organizations was often proprietary\(^{32}\), the complexity and the cost of establishing and maintaining cross-organizational data flows soared. In response, standardization efforts have been made to simplify cross-organizational communication which resulted in the emergence of the Service Oriented Architecture. This led to much looser coupling between the interacting applications of distinct organizations. Thus, cross-organizational data flows implemented through services are not bound anymore to a specific technological platform, including the hardware and software. The data exchange protocols themselves are becoming part of the data flows. This evolution resulted into the current situation of cross-enterprise communication in which the majority of cross-organizational data flows are performed using the Extensible Markup Language (XML) through services.

### 4.3 Transactions in cross-organizational interactions

Since cross-organizational systems appear to have similarities with distributed systems, it may seem at first that transactions in cross-organizational interactions could be handled in a manner close to distributed systems. But several important facts invalidate this hypothesis.

In the case of the alliance situation described in section 4.1, since the information processes are delegated to a dedicated shared information system independently managed by another organization, a company can implement whatever transaction strategy it desires. If the dedicated system itself is a distributed system, a usual transaction strategy with a two-phase commit can be used as described previously.

In case of the archipelago situation, there can't be any transaction outside of a single interaction. This is because there would be an asymmetry between the management's responsibility and the risk mitigation possibilities the management has. The management of an organization providing remote services to different client organizations or companies will never accept to be held accountable for problems it can not control. This would be the case if a client organization could initiate a transaction for a system owned by another organization and if the client organization would neither commit nor roll back the transaction\(^{33}\). Thus, as described in [93], an organization will never allow cross-organizational transactions because of the risk it has of having its resources blocked indefinitely by another organization.

Nevertheless, if multi-interaction transactions are still absolutely necessary, transactional boundaries must be defined between the distributed transactional

---

32 Because of the multiplicity of technology stacks and lack of standardization.
33 For whatever reason: local crash, network problems, etc.
monitor of the service caller and the local transactional monitor of the service provider. If a transaction takes more than a certain amount of time to complete, then there must be a timeout mechanism that cancels the transactions, performs the rollbacks and frees the blocked resources for further transactions.

Although multi-interaction transactions initiated by the client are usually not permitted, this does not mean that single interactions are not transactional. It may well be the case that a single interaction initiated by the client induces some form of transaction on the service provider side. This single interaction initiated by the client can start a transaction which will always either commit or be rolled back independently of any other interaction made by the client. The transaction is always under the sole control of the service provider.

### 4.4 Cross-organizational conversations

Many integrations between organizations require a more *conversational* [111] type of communication. This is implemented through a series of single interactions that may often start and either commit or roll back at the end of each interaction. At the end of an interaction, the service provider sends back some information that is used or analyzed by the client to decide the next interaction to be initiated [111]. These interactions are mainly implemented through *services*.

The service provider may model the whole *conversation* ([111], [112]) as a state machine. The interactions, which can be of transactional nature on the service provider's side, move the conversation from one state to another. At any state of the conversation, there is no risk for the service provider to have its resources technically blocked by a single client\(^3\) [111]. The status of data resources changes as the conversation takes place. Data resources are often given some reserved status until a conversation reaches a point where a confirmation or cancellation interaction is triggered. In case a conversation does not reach an endpoint within a certain timeframe, automatic *compensation actions* ([111], [93]) are taken to reset the availability of the reserved data resources. Compensation actions can also be explicitly initiated by the client to reset the conversation to the start or to any point authorized by the service provider. Thus, cross-organizational transactions and cross-organizational distributed transactions are not used in practice. As shown in Figure 75, cross-organizational process integration is implemented through conversations that use short-lived transactional interactions and compensation actions.

---

\(^3\) By opposition to a cross-organizational distributed transaction.
This means that for process integration involving more than two organizations, it is the client organization that bears the responsibility of driving the process, verifying if all conversations have ended in the desired way, and take compensation actions if necessary with its counterparties [113]. The client organization is the only involved actor that can have a global view through all the involved systems. The client organization is not responsible for maintaining a consistent state within each involved system, but it is the sole responsible for the consistency and the state of the global business process. Cross-organizational interactions are in fact state transitions in workflows. A global cross-organizational business process can involve many different unrelated organizations that may or may not know about each other even if they are all participating in the same business process as shown in the example of Figure 76.
These organizations do not know of each other although they are part of the same business process initiated by the client organization.

Figure 76. Global cross-organizational business process
4.5 Service contracts

As mentioned in section 4.4, cross-organizational interaction approaches are often implemented through services which are used in conversations. In this context, an organization, the service provider, offers a service to other organizations, the service consumers. The relationship between the two parties induced by the service is governed by a service contract\(^{35}\) ([115], [116], [117], [118]).

In this context, a service contract formalizes not only the rights and obligations of each involved party at the business level, but also many other aspects of the technical interaction. Thus, a service contract will often contain terms (or properties) ([116], [117]) covering the functional and non-functional aspects of the relationship:

- How the service is invoked\(^{36}\).
- What does the service return.
- What is the availability of the service.
- What is the type of Quality of Service (QoS).
- How the QoS of the service is monitored.
- How are the security policies enforced.
- etc.

Some of these properties may be fixed unilaterally by the service provider but the contract will usually also specify a set of negotiable parameters [118], such as the cost of the service or the potential penalties, that will be agreed upon by all the contract's parties. Since the relationship between a service provider and a service consumer is different for each provider-consumer tuple, the negotiable parameters may be different for each contract instance. Hence, there may be many contracts for the same service [116].

A service contract can include a very wide range of properties. But as the service contract covers more aspects of a relationship, it becomes more complex to enact and to monitor all properties of the contract. This has led many small or medium enterprises, that offer services, to implement only a subset of a complete Service-Oriented Architecture approach [75].

4.5.1 Existing contract-based approaches

Since the economic implications of service contracting are very important, all these aspects have been extensively researched and many different services contracting approaches ([119], [120], [121], [122], [123], [124]) have been proposed.

---

35 There has also been some research on contracts for objects and components [114], but it has been relatively marginal with respect to the research on service contracts.

36 The service interface
4.5.1.1 Crossflow
The Crossflow approach [119] is based on a dynamic service (process) consumer/provider paradigm. A service marketplace is used by service providers to advertise their offerings. Service consumers use this marketplace to find a service matching their requirements. Service providers and consumers can formally describe the offered and requested services using an XML-based contract specification language. Thus, the relationship between service consumer and providers is managed through contracts. The service provider offers a contract template. A contract includes the service identifier, the required parameters and also a specification of the offered service process. This process specification is described at an abstract level and contains the different activities and transitions of the process. The contract instances are enacted through a service enactment infrastructure that is set up on both provider and consumer sides. Cooperation Support Services (CSS) address the areas of process control and transaction, quality of service and flexible change control that permits the execution of an alternative process path depending of the workflow state and surrounding conditions. The CrossFlow project has evolved into the CrossWorks project in 2004.

4.5.1.2 Contract-based Adaptive Software Architecture (CASA)
The Contract-based Adaptive Software Architecture (CASA) framework [120] focuses on adaptive behavior of applications in mobile environments through the heuristic usage of different software component configurations. The selection of the component configurations is based on the execution environment conditions. There are application-level and component-level contracts. Application-level contracts are dynamically defined between applications providing and consuming services, while component-level contracts are statically specified and are used off-line to help developers compose the behavior alternatives. Each application possesses a Service Negotiator component, which handles the application's service offerings as a provider and service requests as a consumer. The Contract-based Adaptation System monitors the environment resources necessary for satisfying current contracts by querying the Contract Enforcement System and decides whether the component configuration must be changed or not depending on the available resources and the current contract instance specifications. The contracts are specified in an XML-based Contract Specification language.

4.5.1.3 ADEPT
The ADEPT ([121], [122]) architecture uses autonomous agents for negotiating the services that are managed by each of them. The agents negotiate Service Level Agreements\(^\text{37}\) (SLA's) containing the terms and conditions of a service execution. These SLA's are derived from usual business transaction contracts. To overcome

---

\(^{37}\) Contracts.
common limitations in negotiation models, ADEPT has developed a model for evaluating offers and counter-offers. Agents in ADEPT use a predictive model based on the other agents' behaviour and preferences to evaluate, accept or reject an offer or to propose counter-offers. An ADEPT agent will try to find the SLA on a competitive basis that corresponds the most to its own constraints. The evaluation of the different alternatives is performed using a utility function.

4.5.1.4 Federation Contracts

The Federation Contracts approach presented in [123] proposes the use of platform-independent contracts for inter-organizational application management. A federation contract is described in platform-independent terms and contains a business architecture description. The contract also contains the references to the different components used for the inter-application cooperation. Service evolution is mentioned and it is suggested that it should be solved using several component versions simultaneously. It is also stated that, due to the frequent modifications of components, it should be possible to change the contract during the application's lifetime. These concepts are implemented in the PILARCOS prototype.

4.5.1.5 Contract-Based Outsourcing Pattern

The Contract-Based Outsourcing Pattern described in [124] depicts the situation where a service provider outsources partly or entirely the service it is offering to clients. In that situation's global context, outsourcing is viewed as cross-organizational service composition. The authors point out the inherent dependencies between the service requester, the global service provider (the delegator) that is outsourcing a service, and the service sub-providers (the delegates) that are furnishing the outsourced service. It is mentioned that the delegators and delegates take care of their conjoint responsibilities through bilateral contracts negotiated on a bilateral or multilateral basis. There is a contract between the service requestor and the service delegator and a contract between the service delegator and a service delegatee. A contract contains a description of the interactions taking place during the service execution. The contract instances are then fed into contract enactment modules to monitor the correct execution of the service. The contract instances are not built from scratch, but derived from a contract template containing some variable parameters. Service providers offer contract templates either directly to requesters, or they advertise their services on a service marketplace. In the case of a delagator outsourcing a process, both the service requester for the global service and the service delegator will use the marketplace to find the desired service. Contract templates may or may not contain some negotiation possibilities depending on the different actors and the outsourced service. They don't exclude human negotiation but notice that even in this case, the negotiation process can be facilitated through the use of software tools.
5 Dynamic reconfiguration in a cross-organizational context

The set of cross-organizational interactions between IT systems of multiple organizations present a lot of similarities to the interactions between the software elements of a distributed system\(^{38}\). In this context, the applications\(^{39}\) of organizations involved in cross-organizational interactions can be compared to the nodes of a distributed system. Hence, the set of organizations exchanging data can be compared to a *huge distributed system* as shown in Figure 77. Since cross-organizational interactions are implemented through services, dynamic reconfiguration of these interactions is essentially the management of service evolution. Further, if dynamic reconfiguration techniques of distributed systems have proven to be useful, they may be of some help to manage cross-organizational service evolution. Thus, it is important to determine if dynamic reconfiguration techniques of distributed systems are applicable to the evolution of cross-organizational services.

![Figure 77. Organizations as a distributed system](image)

To do this, it is necessary to *compare* the characteristics of distributed systems and cross-organizational interactions. This will be done in section 5.1. Following this, it will be explained in section 5.2 why dynamic reconfiguration techniques of distributed systems *are not suited* for cross-organizational interactions. It will then be described in section 5.3 that even though these reconfiguration techniques can not be directly reused in the cross-organizational context, there are still some reconfiguration approaches that can inspire service evolution management. I will then describe in section 5.4 the real nature of the service evolution challenge.

---

38 As the parallel already drawn for transactions in section 4.3.
39 More broadly information systems.
5.1 Distributed systems in a cross-organizational context

From chapters 2, 3 and 4, we start to distinguish the major differences between distributed systems and systems involved in cross-organizational interactions. As shown in Figure 78, the organizational scope, the type of computation progress unit, the ownership of authority, the technologies used for system implementations and the life cycle or change management show some major differences. Further, it is worth mentioning that security management differs completely in both cases.

<table>
<thead>
<tr>
<th>Distributed systems</th>
<th>Cross-organizational interactions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scope</strong></td>
<td></td>
</tr>
<tr>
<td>Single organization</td>
<td>Multiple organizations</td>
</tr>
<tr>
<td><strong>Computation progress unit</strong></td>
<td></td>
</tr>
<tr>
<td>Transactions</td>
<td>Conversation states</td>
</tr>
<tr>
<td><strong>Authority</strong></td>
<td></td>
</tr>
<tr>
<td>Unique</td>
<td>Asymmetric</td>
</tr>
<tr>
<td><strong>Implementation technology</strong></td>
<td></td>
</tr>
<tr>
<td>Homogeneous</td>
<td>Any</td>
</tr>
<tr>
<td><strong>Life cycle / change management</strong></td>
<td></td>
</tr>
<tr>
<td>Top-down planned</td>
<td>Unpredictable and chaotic</td>
</tr>
</tbody>
</table>

Figure 78. Distributed systems vs. cross-organizational interactions

On one side, the life cycle dynamics of distributed systems can be assimilated to the creation and refinement of high precision mechanical devices. To create them, there needs to be an initial design that is created, implemented, tested and deployed. Then, bug fixing, improvements and changes are carefully planned, realized and deployed. The whole process is usually running under a well-defined leadership and authority.

On the other side, the life cycle dynamics of cross-organizational interactions can be compared more to an evolving living system. Multiple and concurrent changes are made throughout the systems of all involved organizations. There is no guarantee for an organization that its business processes that are spawning over several other organizations will not be short-lived. For any organization, changes in its business domain induce changes within the organization itself. This may lead to necessary changes in the organization's applications and consequently to changes in the interfaces the organization exposes to its partners to automate data flows. Since every organization is subject to this phenomenon, each existing organization may influence its direct partners. Thus, cross-organizational interactions are not stable by nature. The efforts that an organization must sustain for not suffering disruptions due to its evolving world will be strongly correlated to the pace of evolution of the organization's industry and the industries of its partners.
5.2 Inadequacy of distributed systems in a cross-organizational context

Following the descriptions of distributed systems in chapter 2, dynamic reconfiguration of distributed systems in chapter 3, and cross-organizational interactions in chapter 4, it is now possible to determine why distributed systems are not used for cross-organizational interactions and why dynamic reconfiguration techniques used in distributed systems are not applicable in a cross-organizational context. The reasons are the following:

- Heterogeneous products and technologies.
- Authority, responsibility and accountability.

5.2.1 Heterogeneous products and technologies

Dynamic reconfiguration can't be applied in a cross-organizational context because the spectrum of technologies used across all enterprises is much too wide. Each organization has its own subset of information systems, whether they are bought or developed in-house. For using distributed systems in a cross-organizational context, the involved organizations would have to use compatible technologies. This is generally not the case. To still be able to interact, organizations have therefore relied on services, whose usage is independent of the underlying technologies in which they are built.

5.2.2 Authority, responsibility and accountability

Dynamic reconfiguration approaches are not applicable in a cross-organizational context because there is no single organizational authority, responsibility and accountability.

To understand why, lets imagine that the situation mentioned in section 5.2.1 is not true. Lets suppose that information technology products are perfectly standardized in such a way that cross-organizational dynamic reconfiguration becomes technologically feasible. If this improbable case would occur, and whatever the perfection of the implied technology, there would still never be dynamic reconfiguration in cross-organizational processes.

There exists distributed system approaches with dynamic reconfiguration features as described in chapter 3. Even if they have various technical constraints and limitations, they can provide very efficient solutions to particular situations and some of them are on the way to be standardized [125]. Nevertheless, all these approaches are well suited to environments in which there is a single organizational authority that controls and synchronizes the software evolution for

---

40 This would mean that there would exist some type of fits-them-all technology, which is not the case.
all participating software elements.

**Definition – Sovereign system**

A sovereign system [123] is defined as a system controlled by a single authority, for which decisions like application-level operational policies, platform architecture, object models, authorization policies and communication protocols can be made independently from other systems.

Unfortunately, all the dynamic reconfiguration approaches described in chapter 3 are unsuitable when multiple sovereign systems are involved in a network of software interdependencies. The reason for this is that when an application consumes a service from another application, an upgrade of the server application leads to the upgrade of the client application. However, if the client and server applications belong to two different sovereign systems, there is no way to centrally coordinate the upgrades.

In fact, no single authority could unilaterally decide when both component sides should be upgraded, since one of the authorities will never delegate\(^\text{41}\) the technical control of its own applications to the other authority [125].

Hence, even if there would exist a technological unity across all systems of a cross-organizational process, there would never be an automatic upgrade\(^\text{42}\) because of the different chains of responsibilities. Consequently, automatic upgrade\(^\text{42}\) of the client applications when the server applications are modified does not take place in a cross-organizational context. This could only be the case if all applications belong to a single sovereign system.

---

41 Like for resource locking in distributed transactions.
42 Automatic dynamic reconfiguration
5.3 Lessons learned from dynamic reconfiguration of distributed systems

Having determined that dynamic reconfiguration techniques are not applicable in a cross-organizational context may be disappointing but this doesn't mean that every part of dynamic reconfiguration is useless for improving cross-organizational interaction management. In fact, many different aspects of the dynamic reconfiguration approaches of distributed systems are worth being investigated in more depth. Even if these concepts and techniques are not directly applicable, they may be very valuable to minimize disruptions of cross-organizational interactions.

In particular, almost all distributed system approaches stumble on the temporal coordination necessary for the upgrades of the different parts of a distributed system. Some of the existing dynamic reconfiguration approaches ([91], [94], [95]) are built around the concept of using multiple simultaneous versions to be able to relax the constraint of upgrading all elements simultaneously. In these approaches, it is not necessary to upgrade all elements of the system in an atomic operation. These multiple simultaneous versions permit computational threads to continue to run during the upgrade operations as if the system did not change. It is to note that in some of the approaches, it is only the perception of the system by the running threads that is remaining steady while the new versions are deployed [91]. But in the other approaches ([94], [95]), the threads are really running the code as it was deployed when the distributed computation started43.

All the reviewed approaches with such multiple simultaneous versions have correctly identified that synchronizing the upgrade of all elements is a key problem. The higher the number of nodes in a system, the bigger the difficulty of synchronizing them for an upgrade. This is even more true when the different parts of an upgrade do not belong to the same sovereign system.

Since there is no single organizational authority that controls and synchronizes the changes of the different software elements used in the data exchange between multiple interacting organizations44, there is definitely the need for these organizations to agree with each other about how these changes will be made. Thus, as described in section 4.5, it is essential to specify the elements of a service's life-cycle in the service contract established by organizations involved in cross-organizational interactions.

43 This approach in ([94],[95]) permits to cross versions in future and past directions.
44 Which can be considered as sovereign systems
5.4 The service evolution challenge

5.4.1 As defined by researchers

Cross-organizational interactions are implemented using services. The set of changes that these services undergo through time is named *service evolution*. Service evolution is defined in ([116], [126]) as the following:

"Service evolution is the continuous process of development of a service through a series of consistent and unambiguous changes. Service technologies automate business processes and change as those processes respond to changing consumer, competitive, and regulatory demands. Services are thus subject to constant adaptation and variation adding new business rules and regulations, types of business-related events, operations and so forth."

Services undergo different types of change. They include [116]:

- Structural changes.
- Business protocol changes.
- Policy induced changes.
- Operational behaviour changes.

Service changes are also categorized into *shallow* and *deep* changes ([126], [127], [128]). Shallow changes are defined as affecting only the service itself. Deep changes are defined as changes that affect a complete business process in a cascading way beyond the direct clients of the evolving services. As services evolve, the contracts used to formalize the relationship between the service provider and the service clients also evolve. A model for contractual evolution and compatibility is described in [127].

As for the dynamic reconfiguration of distributed system approaches presented in chapter 3, service evolution\(^{45}\) needs to rest on multiple simultaneous versions and backward compatibility to be non-disruptive [128]. The idea of creating upward compatible versions is meant to provide evolution without disrupting existing clients.

\(^{45}\) For shallow changes only, deep changes being out of scope.
5.4.2 Summary

Before defining the real nature of the service evolution challenge, it is worth to summarize the different steps that led us up to this point. As presented in chapter 4, cross-organizational interactions are implemented through services. The similarities and differences between cross-organizational interactions and distributed systems have been highlighted in section 5.1. The inadequacy of distributed systems dynamic reconfiguration techniques in the cross-organizational context has been discussed in section 5.2. In particular, it has been explained in section 5.2.2 that several sovereign systems will never authorize each other to perform upgrades on their respective systems because of their different responsibility chains. Findings from the dynamic reconfiguration approaches applicable to the cross-organizational context have been listed in section 5.3. Service evolution as defined by researchers is described in section 5.4.1. Considering all these facts, it is important to assess the real nature of the service evolution challenge.

5.4.3 The real nature of the service evolution challenge

Organizations interact through services. As business processes change, so do the services used among interacting organizations. No automatic adaptation will ever take place because of the different responsibility chains. Therefore, as service evolution occurs, organizations must devote resources to adapt their applications that use the remote evolving services such that they can continue to interact with their partners. These adaptations must be performed in a timely manner. This means that the applications must be adapted before a given service or service version is not available anymore. Each application using an evolving service will need to be adapted to the new service version. These adaptations will take a certain amount of time.

If an organization does not adapt its applications, it will suffer disruptions due solely to the service evolution of other organizations. Thus, to avoid these disruptions, it is imperative for organizations to provision enough resources to perform the necessary adaptations to their applications. The real nature of the service evolution challenge will now be defined and is the core statement of this thesis.

---

46 Even if a new service version is upwards compatible, the adaptation of applications or the choice of using the new service version will never be taken automatically, but will be always first review by the manager that is responsible for the application.
The service evolution challenge is to determine how many resources an organization needs, and also when these resources will be needed in the future, to adapt its applications that use evolving services from other organizations. These adaptations should be performed in a timely manner such that applications using these evolving services continue to function without any disruptions. If the disruption-free target is not achievable, the adaptations should be done in such a way that the consequences of these disruptions are minimized.

The core statement also induces the following corollary:

In an cooperative world of sovereign systems, planning and resource allocation are the key drivers behind the continuous and smooth operations of organizations. Technology is secondary.

Thus, it is necessary to determine how much resources an organization needs to adapt to its evolving world and also when these resources are needed. Thus, we propose a model that can achieve this goal under certain conditions and hypotheses.
6 Problem description and model proposal

In this chapter, we will describe in detail the problem to be solved and we will propose a model to determine how many resources an organization needs to adapt to its evolving world and also when these resources are needed.

6.1 Definitions

Definition - Exogeneous service

An exogeneous service is defined as a service used by a sovereign system but that is provided by another sovereign system.

Definition - Service evolution event

A service evolution event is defined as any change to a service, whether it is semantic or syntactic, and that can be perceived by a client using the service. This includes changes either in the location of the service, or in the service interface, the service response, the service availability, etc.

Definition - Evolution shock

An evolution shock is defined as the sum of all resources that must be spent by an organization to prevent the disruption of its operations when an exogeneous service used by the organization is subject to an evolution event.

Any organization using services from other organizations will suffer evolution shocks and will need to dedicate some of its resources to adaptation tasks such that it is able to continue its current operations. Thus, evolution shocks lead to evolutive maintenance to avoid disruption. This diverts some resources from the organization that could be used for expanding its operations.
6.2 Problem description

Before defining such a model, it is imperative to understand what are the current problems faced by an organization with respect to the evolution of the exogeneous services\textsuperscript{47} it uses and that are provided by other organizations. Let's assume an organization O has one application. The organization O has secured the availability of an exogeneous service S by entering a service contract\textsuperscript{48} with the service provider of S as shown in Figure 79.

![Simple service](image)

**Figure 79. Simple service**

The exogeneous service S life cycle is not controlled by the organization O. Thus, the service provider of S can decide to phase out the current version of S at any time. When the service provider effectively decides to phase out the current version of the service S, it informs its clients that the service is being phased out. This event is called the termination warning. The termination warning will be reviewed in detail in section 7.3.

When this happens, the service provider can decide to replace the current version of S by a new version that may be upwards compatible or not. In any case, the current service version of S will not be available any more for organization O in the future. Because the abrupt termination of a service would hurt the operations of organization O, the service should remain available a certain amount of time after the termination warning took place. This time period between the termination warning and the effective termination is defined as the termination period as shown in Figure 80. The termination period will be reviewed in detail in section 7.3. It is usually specified in the service contract.

\textsuperscript{47} As defined in section 6.1.

\textsuperscript{48} As defined in section 4.5.
The organization O will have at most a time period equal to the contractually specified termination period to adapt its application using the service S to the new service version or to another service providing the same functionality. The time period that is effectively necessary to adapt the application will depend on the workload\(^{49}\) required to adapt the application and on the amount of resources\(^{50}\) allocated to perform these adaptations. If the organization O allocates more resources to adapt the application, the time period necessary to adapt the application will also be shorter as shown in Figure 81.

The question is now to determine how many resources must be provisioned in advance by the organization O to be ready to adapt the application when the service evolves. This is because it is not possible to recruit immediately resources and have them adapt the application. Moreover, adapting an application requires some knowledge of how the application is structured and functioning. Hence, it is imperative to have resources that possess some insight of the application before they can perform any useful adaptation work\(^{51}\) [129]. Therefore, it is critical to possess and provision these resources as long as the service S is being used by organization O such that they are ready to intervene when service S evolves.

---

49 The workload is usually estimated in man-days
50 i.e. developers.
51 "Effective resource scheduling should ensure that assigned resources have the capability and capacity to execute their assigned tasks, ..." [129]
The type and scale of service S's future evolution are not known at the time the organization should do the resource provisioning. Thus, it is difficult to estimate how many resources the organization O should provision for adapting the application to the evolution of service S. Similarly, the organization O does not know when the evolution of S will effectively happen. Hence, without information about the adaptation effort induced by the future changes and without knowledge on the time at which these changes may happen, it is difficult to provision resources for evolution of service S. The organization O must then choose to follow a certain provisioning strategy. The chosen strategy may vary but would be any mix between the two following ones.

**Strategy 1**

In this strategy, the organization O may follow a very conservative approach. It could provision enough resources continuously during all the time the service S is used. This must be done in such a way that the application can always be adapted immediately in case the service S evolves. Thus, this implies that the organization O always provisions enough resources to be able to handle the worst case scenario. Hence, the organization O assumes that the future evolution of service S will always be the one inducing the most adaptation workload and provisions resources accordingly. If this strategy is pursued, the organization O must train enough resources such that they have enough knowledge about the application. This is not a one-shot effort, but a continuous one, because technically-oriented information technology staff has high turnover rates ([130], [131]). Thus, the organization O must continuously train staff [131] to be able to perform the required adaptations when the service S evolves. Following this strategy, the organization O will incur some continuous costs because of the continuous training. This training will reduce the time to perform the adaptations and therefore lower costs\(^{52}\) due to the adaptations as shown in Figure 82. The adaptation costs equal the costs of all diverted resources from other projects and their consequences [129] to perform the adaptations.

**Strategy 2**

In this strategy, the organization O may not provision any resources in advance at all and wait for an evolution event\(^{53}\) to happen before allocating an amount of resources for the evolutive maintenance of the application. Thus, the organization O would not keep training resources continuously, which would lead to longer adaptation times in case of an evolution event. This is because the resources that will perform the adaptations will not have as much knowledge on the application as they would if continuously trained. When following this strategy, the organization O will incur no continuous training costs but higher adaptation costs because the intervening staff will take longer to perform the changes on the application as shown in Figure 82.

\(^{52}\) Relative to the adaptation costs if the staff is not trained.

\(^{53}\) As defined in section 6.1.
Thus, if the organization O uses a very limited number of exogeneous services, it could follow any of the two strategies. Following the first strategy, the organization O would endure with certainty the fixed continuous training costs but would have potentially lower adaptation costs because of shorter adaptation interventions. Following the second strategy, the organization would not endure any fixed training costs but would suffer higher adaptation costs because of the longer adaptation work as shown in Figure 82.

If however many exogeneous services are used, none of the two strategies is applicable any more. If the organization O embraces the first strategy, it will have to provision enough resources to handle the application's adaptation for each service evolution. This is necessary because the used services could in the worst case evolve simultaneously. Thus, the organization O must be able to handle this case. This means that the amount of resources that must be provisioned to pursue this strategy will increase as the number of used exogeneous services increases. Thus, the fixed costs soar too quickly for that strategy to be followed by the organization's management as shown in Figure 83.
If the organization O follows the second strategy when using multiple exogeneous services, it may not have enough time to adapt its application as the number of used exogeneous services increases. The organization O may be able to allocate some resources to handle the workload of the first few service evolutions, but as more services evolve, the organization will, at one point in time, be short of resources capable of performing adaptations to its application before the used services are effectively phased out. It will need to reuse its resources to perform the adaptations in a sequential order. The waiting adaptation workload will increase as the number of evolving services increases. Thus, the organization O will not only have higher adaptation costs because of the longer adaptation times, but it will also suffer disruptions if it is not able to adapt its application in time as shown in Figure 84. These disruptions also induce costs because the application will not function for some time.
Therefore, none of the strategies are well-suited for handling evolution of multiple exogeneous services, but a combination of both strategies may offer a solution. The basic idea is to provision resources in such a way that all the adaptation work can be done in time without having to provision resources for all of the services. Thus we propose to use a probabilistic approach to estimate the amount of resources that are needed by an organization to handle, with minimum disruptions, the evolution of used exogeneous services. This will be described in detail in chapter 7.

### 6.3 Proposal

To help to solve the previously described problem, we propose a model that, under certain hypotheses, can calculate the amount of resources needed by an organization to adapt its applications using evolving exogeneous services.

The model will do the following:

- Estimate the probability of disruptions induced by evolution of used exogeneous services.
- Estimate the probable resource shortage to avoid disruptions induced by evolution of used exogeneous services.
- Identify the amount of resource shortage.
- Estimate the probable cost induced by the resource shortage.
- Estimate if the probable resource shortage is situational or structural.

---

54 Because of an unfortunate conjunction of events.
7 Definitions, concepts, notions and prerequisites
Before presenting the model in chapter 8, it is first necessary to define some terms, explain some concepts, detail some notions and express some prerequisites.

7.1 Prerequisite: Dependency graph
Dependencies in software engineering have been extensively researched. An attempt to formalize the definition and the characterization of a dependency (in the most general sense) is presented in [132]. A dependency graph based on dependency descriptions is presented in [133]. An extension to dependency graphs targeted to software developers and denominated Annotated Dependency Graphs has been described in [134]. Application Service dependencies have been classified in various groups in [135]. The notion of service dependencies has been used to model web service semantics [136] and web services based process flow composition and service discovery [137]. Dependencies within component-based distributed systems have been presented in [138]. Dependency classification of distributed components offering services has been described in [139]. Analysis of dependencies in component-based systems through a matrix-based approach has been proposed in [140]. A more complete modelling technique that "includes the graphical description of the network and its dynamic reconfiguration as well as the component interfaces and local object systems and their behaviour" is presented in [141]. A model for extracting automatically dependencies from existing networks flows is presented in [142]

Definition – Dependency

In our context, an application A is dependent of an application B, if for A to work properly, B needs to work properly.

There are several types of dependencies for the applications of an organization. At the application level, we distinguish two types of dependencies: intra-organizational dependencies and cross-organizational dependencies.
7.1.1 Intra-organizational dependencies

Definition - Intra-organizational dependencies

Intra-organizational dependencies are defined as the dependencies between applications within a single sovereign system (organization) as shown in Figure 85.

With respect to the organization's size, corporate culture, structure and number of internal applications, the intra-organizational dependencies can be managed through many ways. This will all depend on the structure of the organization's management. In case there is a single hierarchical authority structure, the global set of internal applications can be managed as if it were a huge sovereign system.

This theoretical approach is usually not well suited for real-world situations. When the number of managed systems exceeds a certain limit (empirically determined by each organization), there tends to be some sort of decentralization movement that includes some authority delegation over a subset of the applications. Within a single organization, the evolution of inter-application dependencies may be handled through dynamic reconfiguration of distributed systems techniques as described previously in chapter 3. For this to be successful, the dependency graph between the applications must be known.

The intra-organizational dependency level describes the dependencies between the used exogeneous services and the provided services of an organization. This information is internal to the organization. An application can use an exogeneous service and be itself an element of a dependent chain of internal applications. Some of these dependent internal applications may offer themselves
cross-organizational services to applications that are outside of the organization's boundaries. As shown in Figure 86, the dependency area of interest of an organization includes:

- the dependencies to the exogeneous services provided by other organizations;
- the dependencies between the exogeneous services and the internal applications;
- the dependencies among the internal applications;
- the dependencies between the applications and the provided services to other organizations;
- the dependencies of other organizations on the organization's provided services.

Therefore, there is a need to model the intra-organizational dependencies between the used exogeneous service and the provided services of an organization.

![Figure 86. Dependencies between used exogeneous services and provided services](image)

### 7.1.2 Cross-organizational dependencies

**Definition - Cross-organizational dependencies**

Cross-organizational dependencies are defined as the dependencies between applications of different sovereign systems (organizations).

Cross-organizational dependencies are materialized through the exogeneous services that are used by organizations from other organizations. To analyze this cross-organizational network of dependencies, it is important to know if an
organization is a service provider, a service client, or both. In the case an organization acts as a service client and provider, it is necessary to determine the dependencies that might exist between its used exogeneous service and its provided services. There are several granularity levels at which cross-organizational service dependencies can be observed. The broader view presents the dependencies at the inter-organization level as shown in Figure 87. An organization can be viewed as a client/provider element in the global network of offered services. From this perspective, the intra-organization service dependencies are not observable. The dependency between a used exogeneous service and a provided service is not explicitly stated. Therefore it is not possible to draw the global dependency graph between services through multiple organizations.

![Multiple service consumers/providers](image)

Figure 87. Multiple service consumers/providers

Although process transparency is desirable, an organization that publishes information concerning its internal service dependencies could then be exposed to targeted attacks from hackers. This information is therefore never publicly available. Because there is no publicly available intra-organization consumer/provider service dependency information, there could exist some situations where service dependencies between organizations are cyclical. Some research work considers that there cannot be any inter-organizational dependency information available ([143], [144]). Others have attempted to automatically gather this dependency information despite of the heterogeneity of the managed environments [143].
7.1.3 Service dependencies

Definition – Service dependency graph

Consider an organization with a set $S$ of offered and consumed services as shown in Figure 88. Let the binary relation $D$ on $S$ represent the dependency among these services. Then, $xDy$ means: $x$ depends on $y$ where $x, y \in S$. In other words, if we have $xDy$ then $x$ cannot work without $y$. The binary relation $D$ is reflexive and transitive. The dependency graph of a service $x$ is the pair $(S, D)$ such that $xDy$ is true for any $y$ in $S$ as shown in Figure 89.

The binary relation $D$ is not an order relation. It is because, although not desirable, one cannot exclude cyclical dependencies between services especially if the dependencies cross organizations. In the case the dependency graph does not include any alternative paths, all services in $S$ must work for $x$ to work.

![Figure 88. Services and applications of an organization](image)

However, in the case of fault tolerant systems there might be alternative paths in the dependency graph. Then, $x$ may work even if some $y$ in $S$ does not. Due to the fact that dependencies among services in other organizations are unknown as described in section 7.1.2, the representation of the dependency graph of any service inside an organization cannot go beyond its "dependency area of interest".
A

B

C

D

E

Figure 89. Service dependency graph

**Definition - Organization dependency graph**

An organization's dependency graph is the union of the dependency graphs of all the services and applications of the organization.

As of now and for the rest of the document, unless specified differently, the term *dependency graph* refers to an *organization's dependency graph*.

It is essential for an organization to create and maintain its dependency graph because this permits the organization to determine which applications and provided services will be disrupted in case of a non-adaptation of an used exogeneous service.
7.2 Notion: Securing services with contracts

In the cross-organizational context, applications from one organization depend on services offered by applications of other organizations. If disruption is not an option, client applications must secure the availability of exogeneous services. In this case, the provision of these services must be formally defined through service contracts\(^{55}\) ([115], [145]).

To illustrate the need for contractual guarantees in the context of cross-organizational services, we will consider the case of an ideal sovereign system which is perfectly managed, i.e. this ideal sovereign system is fully exempt of any functional failures and its maintenance process is completely under control and permits a smooth online reconfiguration or evolution without any disruption. In the context of cross-organizational services, this ideal sovereign system can be one of the following types:

- Service provider only: The ideal sovereign system only provides cross-organizational services. It is not a client of any other service provider.

- Service client only: The ideal sovereign system is only a client of cross-organizational services. It doesn’t provide any service to a client.

- Service provider and service client: The ideal sovereign system provides cross-organizational services and uses exogeneous services as a client.

7.2.1 Service provider only

We first consider the case where this ideal sovereign system only offers inter-organizational services to client applications of other organizations as shown in Figure 90.

---

55 As described in section 4.5
Since this ideal sovereign system does not use itself any exogeneous services, it will continue to provide its own services to its clients without any interruption as long as there isn't any network failure. However, the clients of this ideal sovereign system need to have some contractual guarantees regarding the service availability since they might be impacted by a potential evolution shock triggered by a service evolution event that took place at the ideal sovereign system.

7.2.2 Service client only

We now consider the case in which the ideal sovereign system is only using exogeneous services and does not provide itself any cross-organizational services to clients as shown in Figure 91. We suppose that some of its critical computations depend on these exogeneous services. We also limit the case by supposing that there are no alternatives to the used exogeneous services, i.e. that each used exogeneous service is provided only by a single service provider.

The correct execution of the ideal sovereign system is only threatened by these exogeneous services. Since these services are the ideal sovereign system's only instability factor, the management of the ideal sovereign system will require some form of guarantee that the used exogeneous services are available and this for a certain amount of time.

---

56 i.e. an organization
7.2.3 Service provider and service client

We finally consider the case where the ideal sovereign system is not only providing services to clients but also using some exogeneous services as shown in Figure 92.

![Diagram illustrating service provider and service client relationships](image)

Figure 92. Service provider and service client

In this case, it is important to know if some services provided by the ideal sovereign system are dependent on its used exogeneous services. If there are some dependencies between the used exogeneous services and the ideal sovereign system's provided services, the ideal sovereign system needs imperatively contractual guarantees from its own service providers since any disruption of the exogeneous services it uses could affect not only the ideal sovereign system's internal operations but also the operations of its own clients.

In all three cases, there is a need for contractual guarantees, whether an organization acts as a service provider, a service client or both. To provide asynchronous evolution, service providers and service consumers must specify the binding terms that will rule not only the use of these services but also their termination. Service contracts should be standardized and formalized such that the information in the contract can be used in a dependency management system along with the intra-organization dependency information. This information will allow to determine the impact of the evolution of a particular exogeneous service. This resulting information is more than critical in case an organization provides some services that rely themselves on some used exogeneous services for completion as described in section 4.5.1.5.

---

57 A dependency management system is used to model dependencies between elements.
7.3 Concept: Temporal availability properties of a service contract

As seen section 4.5, service contracts can cover a broad scope of properties. Among these properties, a contract should state the nature of the service requested\textsuperscript{58} together with the temporal conditions of its availability. For our model, we need to concentrate only on the \textit{temporal availability aspects} of contracts. The way to specify the nature of the services offered/requested goes beyond the scope of the present thesis.

There are several ways a service's temporal conditions could be specified in a contract. Usually, a service contract specifies the service's availability \textit{start time}, but it may also contain a \textit{fixed termination time} or not, as shown in Figure 93.

**Definition – Service start time**

The start time of a service is defined as the precise time from which the service is available to the client. The service start time is specified in the service contract in an absolute way, i.e. a calendar date.

**Definition – Fixed Termination time**

The fixed termination time of a service is defined as the precise time from which the service in its present version will not be available anymore to the client. The fixed termination time is specified in the service contract in an absolute way, i.e. a calendar date.

In case there is a \textit{fixed termination time}, the service's contract should be renewed before expiration time. If the contract is not renewed, the service cannot be used any more and the application using the service will face disruption. The service contract may not be renewed either because the client does not want to continue to

\textsuperscript{58} Syntax and semantics.
use the service, or because the service provider does not want to continue to provide this service to the client. In the latter case, it may be either because the actual service version is phased out and replaced by another version or because the service provider simply does not offer this service any more.

In any case, if the decision to not renew the contract is taken by the service provider, then the risk for the client application to suffer disruptions increases as time approaches the fixed termination time.

To show this, it is necessary to segment the possible futures that can take place at the fixed termination time:

- **The service contract is renewed.**
  In this case, the service contract is renewed for a new period of time with a new fixed termination time. There is no need to adapt the application.

- **The contract is not renewed for whatever reason.**
  In this case there are the following sub-cases:

  - **The application needs the service to function properly.**
    In this case, the application needs to find another service that fulfills at least the actual functional requirements. Thus, the application needs to be adapted to use this new service.

  - **The application does not need the service to function properly.**
    In this case, the application must be adapted not to use the phased-out service any more or it will break.

We can observe that if the decision to not renew the contract is taken by the service provider, then there is always some adaptation work that must be performed on the client application. But this adaptation work must be done before the service is not available any more, i.e. before the fixed termination time. The organization owning the client application must use some resources before the fixed termination time to perform these adaptations. These adaptations require themselves some time to be performed.
Definition – Service's adaptation time

A service's adaptation time is defined as the absolute time that would be necessary for an organization to perform adaptation work on the applications using the service such that these applications continue to function as desired in the case the old version of the service would become contractually unavailable in the future for whatever reason.

The time to adapt is specified usually either in days, weeks or months. The adaptation time can range from a very short period in case of the roll-out of a new version of the service with minor changes to its semantic or its interface, to a much longer period in case the service is simply phased out without replacement by the service provider which forces the client to seek an alternate service.

As time approaches the fixed termination time, the available time for an organization to perform adaptation work diminishes. Thus, the remaining available time for adapting the applications using a soon-to-be-unavailable service may become shorter than the time needed to perform these adaptations. If this happens, the applications will not function properly. Thus, the risk for the client applications to suffer disruptions increases as time approaches the fixed termination time.

In many cases, a service contract does not specify a fixed termination time. This is because the service provider wants to let the contract run as long as possible and also because he does not know itself how long he will be providing the service. Nevertheless, he does not want to bind itself forever. Therefore, instead of specifying a fixed termination time, many service contracts specify instead an undefined termination time along with a termination period.

Definition – Undefined termination time

If a service contract does not specify a moment in time at which the service availability ends, then the service contract is defined has having an undefined termination time.

59 For financial reasons.
**Definition – Termination period**

The termination period of a service is defined as the time period that the service will continue to be fully available starting from the public announcement that the service version is being phased out. The termination period is specified in a unit of time, usually days, weeks or months. After that period the service becomes unavailable.

**Definition – Termination warning**

The termination warning of a service is defined as the public announcement that indicates to the service's clients that the service is being phased out. From the moment at which the termination warning has been raised, the clients can continue to use the service for a time period equal to the termination period. After that period the service becomes unavailable.

A service contract with an *undefined termination time* will lead at some moment in the future to a *termination warning*, as shown in Figure 94. From that moment, the service will stay available for a time period equal to the termination period. The termination period shall permit the service clients to adapt their applications. Usually, the service clients either adapt their applications to a new version of the service offered by the service provider, if the service provider continues to offer the service through a new version, or try to find another service provider offering a service replacement and adapt their applications to that new service.

Hence, an organization *must manage* the contracts it has with its clients and providers to let its applications work seamlessly as well as to provide some window of time for its application to be adapted.

---

60 The termination period for the same service can be different depending on the client. It is not mandatory that all clients of the service possess the same termination period length.
The contractual relationship becomes even more important in the case where a service provider uses himself some exogeneous services from other service providers to offer a specific services with some added value. Since these specific services are also offered through contracts, it is necessary for the service provider of these specific services to know if the temporal conditions of its offered services are compatible with the temporal conditions of the exogenous services on which the offered services rely on. The used exogeneous services are also denominated as the consumed services. Thus, it is necessary to consider the complete dependency chain of the offered services including the used exogeneous or consumed services as shown in Figure 95.

**Definition – Temporal conditions**

The **temporal conditions** of a service are defined as the time conditions that constrain the availability of a service.

For a fixed termination time service, its availability time interval is written $[t_1, t_2]$, where $t_1$ is the start time and $t_2$ the termination time of the availability of the service $x$.

For a service with an undefined termination time but with a defined termination period, its availability time conditions of $x$ are written $[t_1, T]$, where $t_1$ is the start time of the availability of the service and $T$ the termination period. The interval is open on the right because no upper time boundary is fixed until a termination warning is issued.

---

61 In this case one cannot speak of an availability time interval anymore, because the termination time is unknown.
With the possibility of calculating the *temporal conditions* of an offered service, it is then possible to know at any time \( t \) whether an organization offers its services with *adequate* temporal conditions or not.

**Definition – Service with adequate temporal conditions**

A service with *adequate temporal conditions* is defined as a service offered with *temporal conditions* in such a way that they are within the *constrained temporal conditions* deriving from those of the exogeneous services on which the service itself depends on.

As shown in Figure 95, if we denote \([t_{1\text{offered}},t_{2\text{offered}}]\) the offered temporal conditions of service \( p_1 \) and \([t_1,t_2]_{p_1}\) the constrained temporal conditions of service \( p_1 \), it will be said that the service \( p_1 \) has *adequate* temporal conditions only if \((t_{1\text{offered}} \geq t_1)\) and \((t_{2\text{offered}} \leq t_2)\).

### 7.3.1 Service availability conversion specification

Since both types of contracts, the *fixed termination time* and the *undefined termination time*, can be encountered for the offered services, it is very important to determine if it is possible to convert from one contract type into the other one. If this were the case, it would reduce strongly the complexity of the global model that we propose. Thus, further modeling would be simplified.

In the case an organization is both a service provider and a service client, a dependency graph should be used to validate the consistency of availability time defined in the services’ contracts. In fact, the provided service’s availability time must be compatible with the availability time of all services in its dependency graph. To formalize this compatibility, it is necessary to distinguish the services with a *fixed termination time* from the services with an *undefined termination time*. From now on, we will only consider dependency graphs without alternative paths.

Let \( x \) be a service with a fixed termination time with its availability time interval \([t_1,t_2]_x\). Then, the availability of a service \( x \) with a dependency graph \((S,D)\) is given by the highest start time and the lowest termination time among all services of its dependency graph. Let us define the following three functions:

1) \( tc(s) \) returns the temporal conditions of a service \( s \). For a service with fixed termination time we have: \( tc(s) = [t_1,t_2]_s \)
2) \( start([t_1,t_2]_s) = t_1 \)
3) \( end([t_1,t_2]_s) = t_2 \)
As shown in Figure 96, let p be a service that is provided by the organization and (S,D) its dependency graph. Let us have \( \text{START}_s = \{ k \mid \forall s \in S, k = \text{start}(tc(s)) \} \) and \( \text{END}_s = \{ k \mid \forall s \in S, k = \text{end}(tc(s)) \} \). Then the availability time interval of the provided service p is given by:

\[
[t_1, t_2]_p = [\max(\text{START}_s), \min(\text{END}_s)]
\]

Let \( x \) be a service with an undefined termination time but with a defined termination period with availability time conditions\(^{62}\) \([t_1, T_x]\). The availability of a service that depends only on services with defined termination periods is constrained by the highest start time and the smallest termination period among all services of its dependency graph. To formalize this relationship, let us first complete the definition for \( tc(s) \):

4) \( tc(s) \) returns the temporal conditions for a service \( s \). For a service with defined termination period we have: \( tc(s) = [t_1, T_s] \).

As far as contractual availability time of an offered service is concerned, one should always do the worst-case analysis of the prerequisite services’ availability time. For example, if a prerequisite service \( y \) has a termination period of length \( T \), then the shortest termination time at time \( t \) is \( t + T \) unless a termination warning has already been issued. This lets us define a worst-case analysis mapping between the fixed termination time and the defined termination period scenarios.

Let \( s \) be a service, and \([t_1, T_s]\) its temporal conditions. As far as computing the availability time for a consumed service is concerned, at time \( t \) we must consider the following cases:

---

62 In this case one cannot speak of an availability time interval anymore, because the termination time is unknown.
In case no termination warning has yet been issued at time \( t \) for service \( s \), we have the following situations:

- if the service provider of \( s \) doesn’t have the right to send a termination warning before \( t_1 \), we have:
  - a. \( (t < t_1) \implies ([t_1, T_s] \leftrightarrow [t_1, t_1 + T_s]) \)
  - b. \( (t \geq t_1) \implies ([t_1, T_s] \leftrightarrow [t_1, t + T_s]) \)

- if the service provider of \( s \) has the right to send a termination warning before \( t_1 \), we have:
  - c. \( (t < t_1) \implies ([t_1, T_s] \leftrightarrow [t_1, t + T_s]) \) (same as b)
  - d. \( (t \geq t_1) \implies ([t_1, T_s] \leftrightarrow [t_1, t + T_s]) \) (same as b)

In case a termination warning was issued at time \( t - \Delta t \) for service \( s \), we have the following situations:

- if the service provider of \( s \) doesn’t have the right to send a termination warning before \( t_1 \), we have:
  - e. \( (t - \Delta t \geq t_1) \implies ([t_1, T_s] \leftrightarrow [t_1, t - \Delta t + T_s]) \)
  - f. \( (t - \Delta t < t_1) \) cannot happen because the service provider has not the right to do it.

- if the service provider of \( s \) has the right to send a termination warning before \( t_1 \), we have:
  - g. \( (t - \Delta t \geq t_1) \implies ([t_1, T_s] \leftrightarrow [t_1, t - \Delta t + T_s]) \) (same as case e)
  - h. \( (t - \Delta t < t_1) \implies ([t_1, T_s] \leftrightarrow [t_1, t - \Delta t + T_s]) \) (same as case e)

The time \( t \) used for the above relationships can be set arbitrarily. However, one should always compute the availability of services at time \( t = \) (present time) for the following reason. It cannot be known in advance when a service provider will decide to cancel a service. Computing the availability time of an offered service \( p \) that depends on another used service \( s \) without fixed termination time at calculus time \( t > (\text{present time}) \) induces uncertainty. Therefore, for any case with time \( t > (\text{present time}) \), one must consider the worst case analysis as if a termination warning has been issued at \( t = (\text{present time}) \) which restricts the availability time period of \( s \) from \([t_1, T_s] \) to \([t_1, t - \Delta t + T_s] \) with \( \Delta t = t - (\text{present time}) \). This implies that for time \( t > (\text{present time}) \), the length of the availability interval is \([t_1, \text{present time} + T_s] \). For any time \( t \leq (\text{present time}) \), the different cases described above apply. Except for case "a." which leads to a constant interval, all other cases will constrain the availability time interval depending on \( t \). Therefore, for any time \( t < (\text{present time}) \), the availability time interval will be smaller than the one for \( t = (\text{present time}) \). This implies that we should always choose for the computations the time \( t = (\text{present time}) \).

Due to these conversion rules, the computation of the time interval of a service based on the time conditions of its own used services will depend on the moment
in time considered. Let us define the mapping function convert(t, interval) that will convert, at time t, an interval expressed as a time condition into an interval expressed as a fixed termination time:

\[
\text{convert}(t_1, [t_1, t_2]) = [t_1, t_2]_x
\]

- If the interval is already in fixed termination time format (identity function)

\[
\text{convert}(t_1, [t_1, t_2 + T]) = [t_1, t_1 + T]_x
\]

- If no termination warning has yet been issued at time t and the service provider doesn’t have the right to send a termination warning before \(t_1\) and \(t < t_1\)

\[
\text{convert}(t_1, [t_1, t_1 + T]) = [t_1, t_1 + T]_x
\]

- If no termination warning has yet been issued at time t and the service provider has the right to send a termination warning before \(t_1\)

\[
\text{convert}(t_1, [t_1, T]) = [t_1, t_1 + T]_x
\]

- If no termination warning has yet been issued at time t and the service provider doesn’t have the right to send a termination warning before \(t_1\) and \(t \geq t_1\)

\[
\text{convert}(t_1, [t_1, T]) = [t_1, t_1 - \Delta t + T]_x
\]

- If a termination warning was issued at time \(t-\Delta t\)

Based on this function we can reuse the same expression as before to compute the availability time period of a service that is dependent on a graph of used services, provided that we have converted the format of the time intervals as shown in Figure 97.

Let \(p\) be a service that is provided by the organization and \((S, D)\) the dependency graph of \(p\). Let us have \(\text{START}_s = \{k \mid \forall s \in S, k = \text{start}(\text{convert}(t, tc(s)))\} \) and
END_s = \{k \mid \forall s \in S, k = \text{end} (\text{convert}(t, \text{tc}(s)))\}. Then the availability time condition of the provided service p is given, as before, by:

\[ [t_1, t_2]_p = \left[ \max(\text{START}_s), \min(\text{END}_s) \right] \]

Finally, since we are able to convert the temporal conditions from one format to another, we can compute the availability period of a service that depends on a set of consumed services that have their temporal conditions represented in both formats as shown in Figure 98. Let p be a service that is provided by the organization and (S,D) its the dependency graph. Since the convert(t,interval) function takes into account both interval formats, we can reuse the same expression as before.

Let us have \( \text{START}_s = \{k \mid \forall s \in S, k = \text{start} (\text{convert}(t, \text{tc}(s)))\} \) and \( \text{END}_s = \{k \mid \forall s \in S, k = \text{end} (\text{convert}(t, \text{tc}(s)))\} \). Then the availability time condition of the provided service p is given by:

\[ [t_1, t_2]_p = \left[ \max(\text{START}_s), \min(\text{END}_s) \right] \]

So far, we only considered the case of dependency graphs without alternative paths. In the case of fail-over mechanisms providing alternative paths in the dependency graph, the computation of the time interval for p must take the topology of the graphs into account. In the case of parallel paths, the time interval to take into account is the largest among the alternative paths. To calculate the time interval of an offered service p, it is then necessary to recursively explore its dependency graph and evaluate at each node its constrained time interval.
7.4 Notion: Service evolution scenarios: from one version to the next one

7.4.1 Unannounced service evolution
In the simplest case, a service provider abruptly changes the service interface and/or semantics without advance notice\(^{63}\) [128] and asks the service clients to adapt after the evolution took place as shown in Figure 99.

![Figure 99. Unannounced service evolution](image)

Hence, the corresponding client applications will break and remain unusable until they are adapted to the new version of the service. It must be noted that we consider the transaction duration to be close to instantaneous with respect to the version lifetime. If this were not the case, the offered service would also create certain inconsistencies at switchover time for long transactions exceeding the switchover time.

7.4.2 Temporal coordination
It may not be possible to perform an automatic reconfiguration of the client and server applications involved in cross-organizational interactions, but it is still necessary to adapt the applications using evolving services because the server applications will continue to evolve asynchronously [123] of the client applications. Therefore, since some client adaptation work must take place, one solution could be to synchronize the participating sovereign systems so that all the changes, on the client and the server applications, are made at the same time. This synchronization is also denominated temporal coordination.

\(^{63}\) As stated in [128]: "New services are created without any notification, providers interrupt the provisioning of services without any indication and the functionality of services changes overtime. Changes can happen at any stage in the service life cycle and have an unpredictable impact on the service stakeholders."
Definition – Temporal coordination

The temporal coordination for a service evolution is defined as the process through which the service provider synchronizes all clients using the evolving service such that they all start using the new version of the service exactly at the same time.

Temporal coordination involves informing the service's clients sufficiently in advance such that all clients can adapt their applications to the new version of the service as shown Figure 100. Thus, the clients have a certain delay to adapt their applications. When the delay expires, the switchover will occur for all service clients, whether they have adapted their applications or not. The clients are not bound to perform the adaptations to their code at the same time. It is only mandatory for all clients to deploy it in production at the same time. The clients that were able to adapt their applications will continue to work with the new version and the others will suffer disruption. Even if the temporal coordination scenario is better than the unannounced service evolution scenario because the clients may avoid disruption if they adapt in time, it still requires all the clients to be coordinated to switchover at the same time.

![Figure 100. Announced service evolution](image-url)
In order to achieve this, the preparation of the upgrade must be carefully planned a long time in advance for all the participants. They need to be informed of the incoming changes and build the corresponding new versions of their applications. This means that the clients using the service must know well in advance of the production deployment time the specification of the new service version. If a termination period has been specified for the service, it prevents the service provider from accepting change requests for that service from its clients and deploying them rapidly. Furthermore, in case an upgrade must be rolled back, it is necessary that all the clients of the upgrade get quickly informed of the situation in order to be able to roll back their own application upgrades themselves. Due to the inherent coordination complexity, the temporal coordination is not scalable beyond a very small number of participants. Hence, as interactions develop between many sovereign systems, this temporal coordination may well become intractable.

It is important to note that the client and server parts of a cross-organizational interaction are usually not stand-alone software elements but can be deeply nested in their enclosing application. The situation is even worse when the client system consumes services from several server sovereign systems. Since each organization providing services has its own upgrade or evolution planning, the probability that a client system must adapt itself to the evolution of more than one server component increases with the number of service providers. If not managed, this could lead to unbearable constraints on the client system development team as the temporal coordination of components upgrade cannot be ensured when several sovereign systems are involved. Below, we identified several scenarios that can occur in the evolution of a service interface or semantic specification when using temporal coordination.

However, temporal coordination can generate evolution management scheduling conflicts. In fact, if one client consumes two services A and B from two different service providers, and if B should be adapted to a new version at a time close to the version change of A, it becomes difficult for the integration team to decide what to adapt. Should it adapt its application to use the new version A at time $T_a$, or should it adapt the application to use the new version of B at time $T_b$? In any case, with limited human resources available, the adaptation must be serialized. Then there might be some switchover deadline trespassing for any of the new versions of services A or B, which will result in application disruption time as shown in Figure 101.
Lets us now consider the situation where, for a given client application, the adaptation to the new versions of A and B is interdependent and the switchover time of A is set before the switchover time of B. In this case, the integration team must prepare two new versions of the client application. The first version, a temporary one, will use the new version of A and the old version of B between A and B switchover time. A second version of the application will use the new version of both components after the switchover time of B.

However, if one takes into account the fact that a version switchover by the server system could fail or be postponed, one must deal with two additional scenarios. The first is when the switchover of the new version of A fails. Then, the client application must be adapted to the new version of B together with the old version of A. The second scenario is when both switchovers fail. The client application will then continue to run its old version. The case where the switchover of B fails is the same as before, but the temporary version will be kept until the new version of B is ready again.

In the worst situation, to reduce application disruption risk relative to the simultaneous evolution uncertainty of its N used exogenous services, one must generate and test $2^N - 1$ versions of the client application.
7.4.3 Multiple simultaneous versions

As for some dynamic reconfiguration approaches of distributed systems described in chapter 3 which rely on simultaneous component versions, researchers ([123], [126], [128], [63], [91]) in the SOA domain have similarly advocated to use multiple simultaneous versions of services to overcome the difficulties of managing service evolution. One of the key differences between the distributed systems and cross-organizational interactions is the fact that there is not a single organizational authority, responsibility and accountability in the latter case as described in section 5.2.2. Because of this, no automatic upgrade or evolution will ever take place, disregarding any advance in technology as stated in section 5.4.3. Temporal coordination has been mentioned in section 7.4.2 as a possible approach for service evolution, but it has been shown to be inapplicable as the number of services and client rises. This is essentially caused by the synchronicity constraint.

Definition – Synchronicity constraint

The synchronicity constraint is defined as the obligation for all the clients of a service to move to the same service version at the same time.

In the case of multiple simultaneous service versions, the central coordination of the clients from the service provider is not necessary anymore. As shown in Figure 102, there is in fact a time frame during which the client organization will adapt to the new version of the service and make the effective switchover to this new version.

![Figure 102. Asynchronous evolution](image)

Hence, providing multiple simultaneous service versions relaxes the synchronicity constraint among all clients of the evolving services.
7.4.4 Implications of relaxing the synchronicity constraint

It is possible to relax the synchronicity constraint of a service if the service provider keeps several service versions available simultaneously for some period of time. This allows clients to *asynchronously adapt their applications* to the new service versions, each at its own pace.

This does not mean that the service provider must keep as many code versions as there are service versions. In fact, many service versions can even be handled by the same running code. This scheme is nothing more than the case where a server component implementation is presented through several interface versions, themselves implemented by some type of facade components.\(^{64}\) This technique is called the *dynamic request upgrade* in the dynamic reconfiguration approach proposed in \([91]\) as described section 3.6. The *dynamic request upgrade* approach provides evolution capabilities without changing the clients that use previous service interface versions. This allows a service interface to evolve not only on a type-subtype basis, but also to a new version syntactically incompatible with the old version.\(^{65}\)

We have mentioned in section 7.4.2 that the temporal coordination for service version changes involving several service clients and providers is not a scalable solution. Such a temporal synchronization can be viewed as a form of shared serialization in the evolution path of the provided services. In the case of temporal coordination, the lifecycle management decisions of all clients of a service are driven by the service provider due to the service version changes. This means that the service provider creates a form of indirect temporal dependency between unrelated applications (the service clients) through the temporal synchronization constraint. Ideally, from a client perspective, the exogeneous services of a service provider should remain the same as long as the client needs them. It is important to note that this does not imply that the provider should never *change* his services, but only that each client's *view* of the consumed services should remain the same through time. To achieve this, it is necessary to relax the synchronicity constraint. This means that the service provider should offer some form of evolution mechanism enabling the different versions of the offered service to coexist for a certain period of time.

\[^{64}\text{Which are adapters to the current running implementation.}\]

\[^{65}\text{To achieve this, the model uses mapping operators to chain the invocations of clients on old server types to new server types. These mapping operators are objects that represent the old services and act as mediators between the clients and the new version of the service.}\]

\[^{66}\text{Relative to the number of clients and services.}\]
Concurrent version management is essential to enable the asynchronous evolution of service consumers. It makes it possible for organizations to *manage their software development resources* to adapt the client applications at their own pace.

In fact, this is equivalent to a *serialization shift*. The deserialization of service versions released by the service providers allows the serialization of adaptations by the service clients as shown in Figure 103 and Figure 104.

![Figure 103. Serialized version release](image1)

![Figure 104. Deserialized version release](image2)

In summary, one of the most important factors for avoiding uncertainty related to the evolution of exogeneous services is to have a flexible switchover time from one service version to the next one. Another important factor is the duration of the switchover time interval.

It is to note that the time that is needed to adapt a service should always be significantly shorter than the time the old version of a service is kept available, i.e. the *service adaptation time* should always be shorter than the *termination period*.
7.5 Prerequisite: Information related to future evolutions

One of the key elements to determine how much resources an organization needs to adapt to its evolving world and also when these resources are needed as stated in section 5.4.3 is information related to the expected evolutions of the used exogeneous services. Since this information is not currently provided by the service providers\(^{67}\), it is necessary to collect or derive information about the services’ future evolutions. This information is materialized as a service's evolution probability distribution.

**Definition – Service evolution probability distribution**

The *service evolution probability distribution* is the set of daily probabilities\(^ {68}\) of the issuing of a termination warning for the service. A service evolution probability distribution is meant to cover a relatively foreseeable future period of time during which the service could be subject to an evolution event. The *service evolution probability distribution* can be of any type\(^ {69}\).

\[\text{probability of evolution}\]

\[\text{present} \rightarrow \text{Foreseeable period of time}\]

\[\text{time}\]

**Figure 105. Service evolution probability distribution**

As described in section 7.4, if a service is being terminated, the provider sends a termination warning to all the service's consumers. From the moment the termination warning has been sent, a service consumer uses the termination period specified in its service contract to either adapt to the new service version offered by the same service provider or to find an equivalent service offered by some other service provider and adapt to that one.

---

\(^{67}\) The author was not able to find by any means anything that would be even close to it. It was not even possible to find a reference that would state it could be the case.

\(^{68}\) As estimated by each service consumer.

\(^{69}\) Gaussian, Uniform, Poisson, etc.
7.5.1 Time period length of a service evolution probability distribution

From the professional experience of the author as "Head of Software Development" of a major financial firm, no organization foresees its software development and maintenance\(^{70}\) tasks more than one year in advance. Organizations *do know in advance what* they wish to do with their information systems, whether it is new developments or evolutionary maintenance, but they *cannot precisely plan the timing* of these future tasks beyond a relatively short period of time. This is because of the evolutionary nature of the organization's business and of its partners, which is reflected in the evolution of an organization's exogeneous services. This results in continuously changing software development priorities\(^{71}\) \([129]\). Thus, we suggest that the time period of the service evolution probability distribution *should not exceed one year*.

This does not mean that it is strictly impossible for an organization to plan these evolution maintenance tasks more than one year in advance, but as the estimation is done for a more distant future, the probability that this estimation makes sense diminishes. Moreover, since in our model these probability distributions will be used conjointly to perform calculations, this means that even if one organization has a very clear view on its maintenance tasks, the other organizations may not. Thus, it would make little sense to try to apply our model when only a very small part of the necessary data would be available.

In a similar fashion, it is of little utility to have a too short period of time. There is no added value to have an estimation only for the next five days. Thus, we suggest that the period covers *at least the next three months*.

7.5.2 Collecting service evolution information

The evolution probability distribution of a service can be modeled in different ways. This depends on the possibility to collect information about the expected evolutions of a service from its provider. If it is possible, one approach is to question the provider about the expected evolutions of the service and translate his input into a probability distribution. An automatic data collection proposal of this kind is described in section 7.5.4.3.

In case it is not possible to get any information from the provider, it is necessary to make our own arbitrary assumptions about the distribution. One possibility is to use an heuristic approach based on the life cycle of the service's precedent versions or on the life cycles of all service versions known to the consumer.

Another possibility is that there is no available data about the future evolutions of

---

\(^{70}\) In the sense of evolutionary maintenance, and not corrective maintenance.

\(^{71}\) "But disruptive events such as requirements changes, needs for fixing important bugs, incorrect or unexpected process execution, and staff turnover can create uncertainty that complicates resource scheduling" \([129]\).
the service. In that case, it may be appropriate to set a daily probability for the release of a termination warning that is constant for a defined time interval as shown in Figure 106.

![Figure 106. Equal probability on case no information is available](image)

However, the way to handle this lack of information is not always similar and must therefore be done by an organization on a case-per-case basis.

### 7.5.3 Notion: Aging of information

The information on the evolution probability distribution that has been obtained for a service ages as time passes. Hence, the relevance of this information that has been collected in the past decreases over time. Thus, what was only seen as a probability at the time the estimation was done in the past becomes a certainty as time passes\(^72\). This alters the probabilities of the remaining future that have been estimated in the past.

To illustrate this, let's consider a service S at time \(t\). For this service S, information about its probability of evolution is also collected at time \(t\). The time period covered by the service evolution probability distribution starts at time \(t\) and ends at time \(t_{end}\). As suggested in section 7.5.1, \(t_{end}\) is not bound to be a particular fixed value but it should ideally be greater or equal to 3 months and smaller or equal to \((t + 1 \text{ year})\).

We assume that an evolution event will happen\(^73\) between time \(t\) and \(t_{end}\) but we don't know the exact time of the future event. As shown in Figure 107, the time period covered by the evolution probability distribution ranges from time \(t\) to time \(t_{end}\). In our model, the finest granularity of the collected information is day-based.

\(^{72}\) In the sense that we know if an evolution event took place or not.

\(^{73}\) The sum of all probabilities for the period is equal to 1.
As time $T$ passes, the information related to the evolution probability distribution of the service $S$ for that period $T$, that had been collected at time $t$, becomes irrelevant\(^{74}\) as shown in Figure 108.

This is because there is no uncertainty anymore about the occurrence of an event during a period in the past. Thus, as shown in Figure 109, it is necessary to readjust the probabilities of the remaining period for which the probabilities had been defined at time $t$. If no new information is available at time $t + T$, this means that the estimation made at $t$ is the one that is still valid. Thus, since we had assumed an event would happen between $t$ and $t_{\text{end}}$, and it has not happened between time $t$ and time $t + T$, this means that it must happen between $t + T$ and $t_{\text{end}}$. Thus the sum of all probabilities between $t + T$ and $t_{\text{end}}$ must be equal to 1. Because we do not have more information at $t + T$ on the probabilities of the remaining period between $t + T$ and $t_{\text{end}}$, we scale the probabilities such that the sum of all probabilities between $t + T$ and $t_{\text{end}}$ equals 1. The scaling factor $SF$ that is applied is defined as:

\(^{74}\) This information initially covered the period between time $t$ and time $t_{\text{end}}$
Managing evolution risk of cross-organizational services

\[
SF = \frac{1}{1 - \text{sum(probabilities between } t + T)}
\]

at time \( t \)

end

\( t + T \)

(present)

\( t_{\text{end}} \)

Figure 109. Probabilities adjustment at time \( t + T \)

This must be done each time that the model is calculated. Since the evolution event did not take place between \( t \) and \( t+T \) and since the event is meant to take place during \( t \) and \( t_{\text{end}} \), it is therefore necessary to increase the probabilities of the remaining period such that they keep their relative values one to each other and that the sum of all probabilities is equal to 1.

7.5.3.1 Dangers of information aging

The information aging can also lead to misinterpretation of the results calculated by the model. To illustrate this, let's consider a set of services at a time \( t \). We assume that the information about their evolution probabilities is also collected at time \( t \). As time passes, the model is periodically recalculated. At time \( t+T \), when using the probabilities that had been estimated at time \( t \) and adjusted at time \( t+T \) as shown in Figure 109, the model may provide results that may be wrongly interpreted. The model may suggest that the organization is on the brink of suffering a high amount of disruptions/damages because it will not be able to adapt its evolving exogeneous services in time.

Even if the calculation of the model is technically correct, this alarming result is in fact a false interpretation. If the information is too old, the model will predict such results. This is because if no fresh information has been collected, and if time passes, the probability that an event happens for each day of the remaining time period of the service evolution probability distribution increases. Thus, the results at time \( t+T \) are alarming not because the situation is bad, but because the
information is too old. These results were calculated at \( t+T \) based on the information that was thought to be the most accurate at time \( t \). Since the remaining probabilities have been adjusted\(^\text{75}\) at \( t+T \), there can only be more alarming results as time passes. But at time \( t+T \), this information may be simply outdated and should therefore be revised or recollected. Newly collected probability distributions at \( t+T \) may prove to be completely different than the ones at \( t \) as shown in Figure 110. This is because the service provider may know much better what his service evolution plans are at time \( t+T \) for the period between \( t+T \) and \( t_{\text{end}} \) than at time \( t \). Hence, collecting and using fresh data at time \( t+T \) can change completely the results yielded by the model on the base of the data collected at time \( t \). The results yielded by the model at time \( t+T \) with fresh data represent better the real situation at time \( t+T \).

![Figure 110. Updated information at time \( t+T \)](image)

### 7.5.4 Information updating policy

The relevance of model results depends on the "freshness" of the evolution probability distribution of each service. Thus, our model should be used with "fresh" probability data and not with probability data that have already become irrelevant. Hence, it is necessary to set up a collecting strategy to keep the probability data up-to-date. There are several approaches to determine when to recollect this data.

#### 7.5.4.1 Recollection based on the data age

The first approach consists of defining a maximum authorized age for using collected service evolution probability data. This implies that when the model is used to yield results, all the information related to the evolution probability distributions of all involved services are not older than a certain age.

---

\(^{75}\) They are always increased by definition because of the value of the scaling factor \( SF \) (\( \geq 1 \)).
Definition – Age of service probability distribution

The age of a service probability distribution is defined as the time that has passed since the service probability distribution has been estimated by the service provider.

From a client's perspective, this is not the time of the client's former evolution probability data collection for that service, but it is the effective time since the service provider did the estimation. To be valid, all service evolution probability distributions must have been collected during a time interval with a lower boundary denominated $t_{\text{oldest}}$ as shown in Figure 111. We define $t_{\text{oldest}} = (\text{present time} - \text{maximum authorized age})$. Before using the model, if a client detects that a service probability distribution is older than the maximum, then the client should require and recollect a new estimation of the service probability distribution from the service provider.

![Figure 111. Far horizon estimation](image)

The advantage of this approach is that the age of the information is guaranteed. This implies that an organization must collect information about the evolution probability distributions at least once every "max age" period of time. A disadvantage is that it may not be possible to collect the evolution probability distribution information of all used exogeneous services at client's desired frequency. Another disadvantage is that since all services have different probability distributions with different time horizons, it may happen that a particular service has a probability distribution that is already completely irrelevant, even though the information has been collected within the defined time frame.
7.5.4.2 Collecting data based on the proportion of obsolete data

The second approach consists of collecting an evolution probability distribution of a particular service only when a certain proportion of the daily evolution probabilities estimated at time t have become irrelevant as shown previously in Figure 109. Thus, it is possible to wait until time $T_{76}^7$ before recollecting evolution probability data. This time $T$ is determined after the proportion of irrelevant evolution data reaches a certain predefined threshold. This approach is more sensitive than the previous one to the amount of data that become irrelevant$^{77}$. It is the information itself and not its age that triggers the data recollection.

The advantage of this approach is that it is not necessary to recollect data on a regular basis, but only when a sufficient proportion of data has become obsolete$^{78}$. This approach works well when an evolution event is meant to take place in the close future. In this case, the level of the evolution probability will be greater for the immediate future. Thus, the threshold of obsolete data will be reached much sooner and the evolution probability data will be recollected accordingly.

The disadvantage of this approach is that if the evolution probability distribution set at time t extends to a far time horizon, the service provider may change the service's probability distribution at a time $t+T$ between time t and the upper bound of the covered estimation period. Depending on the shape of the probability distribution, it may be necessary to wait a very long period of time before the predefined threshold of obsolete evolution data is reached and a new probability distribution is recollected by the client as shown in Figure 112. Thus, the model will not use the most up-to-date data and will yield results of poor quality.

![Figure 112. Disadvantage of triggering recollection using a threshold](image)

---

76 The length of this period of time $T$ is undefined at time $t$.
77 Again, because of certainty
78 In the sense that a proportion of the data is not probability anymore but certainty because time has passed and events have not happened.
7.5.4.3 Collecting data automatically on a regular basis

Each of the two previously described approaches have some shortcomings and are not sufficient to cover all cases when used separately. Thus, a combination of both approaches used together could provide some form of data quality for the service evolution probability distributions if the parameters used in each approach are restrictive enough. The issue is that this induces frequent data recollections.

Thus, to solve this issue and to have up-to-date service evolution probability distributions, an additional approach is to define a method to collect the data automatically. If this were the case, data could be collected frequently and on a regular basis. On the opposite, if collecting the evolution probability distributions can't be done automatically, it is very likely that clients will not collect evolution probability distributions often enough for the model to yield significant results because of the required effort. This will lead to increasingly deteriorating results from the model until they reach an alarming level. It is reasonable to believe that because of the effort required by an evolution probability manual recollection, it is only when an alarming level is reached that an organization will try to collect the most up-to-date information. This means the model will not be useful on a day-to-day basis, which it should be. Thus, the collection process of evolution probability distributions should be automated and performed on a regular basis.

7.5.5 Prerequisite: Service availability as a service

In the ideal case, for every offered service, the service provider should provide information on the service's future expected evolutions. This information about the service must be available itself as a service such that the collecting of this information can be done automatically as shown in Figure 113.

![Figure 113. Service evolution information as a service](image)

The information that must be provided comprises the service's evolution probability distribution, the covered time period, and the date on which the distribution was estimated as shown in Figure 114.

---

79 This is if the maximum age tends to 0 in the first approach, and if the threshold of the proportion of obsolete data tends to 0 in the second approach.
7.5.5.1 Service evolution information accuracy

It is to note that since the information about the expected evolution of services is provided by the service provider, its relevance and accuracy is dependent on the service provider only. This means that if the service provider does not provide information with sufficient quality about the future of its services, our model will not be very accurate.

7.5.5.2 Estimating the information quality

The service evolution information is said to be of sufficient quality if the last estimated data is close to the effective evolution event data. This does not mean that estimated data for the future service evolutions must be absolutely perfect a long time in advance. The estimations can change over time but the effective occurrence time of the evolution event should not differ to a large extent from the estimated data. If the estimation is well performed and made available regularly by the service provider, there should not be any important surprise as time passes as shown in Figure 115. This means that as time passes, if the probability distribution is updated regularly, the evolution probability distribution standard deviation should become smaller and the event occurrence time should be close to the mean of the distribution.

But if the available estimation data is of poor quality, the effective event will take place at a different time than the most probable time as projected in the last available estimation made by the service provider as shown in Figure 116. This poor data quality can be the consequence of a bad estimation or because the service provider has not re-estimated regularly the probabilities of an evolution event for the service\textsuperscript{80}.

\textsuperscript{80} The service provider may have always posted the unrevised initial version.
Figure 115. Estimation with sufficient quality
Making available the estimations of service probability distributions forces a service provider to plan and regularly communicate its evolutive maintenance tasks. A service provider could choose not to update this information too frequently, but in this case he will face many requests from clients to update his information about the future evolutions of his services. Thus, if a service client judges that the date of an estimation for a service is not recent enough, the service provider should be able to reassess the probable future evolutions he will roll out for his services. Hence, the service provider should be able to produce an updated estimation on demand and make it available in a short period of time such that its clients can use it. Therefore, the service provider would be often diverted from his own work to respond to these requests from clients.
8 Model

As mentioned in section 5.4.3, the goal is to create a model that can *determine how many resources an organization needs to adapt its applications to the evolving exogeneous services and also when these resources are needed*. We listed the model targets in section 6.3 as the following:

- Estimate the *probability of disruptions* induced by evolution of used exogeneous services.
- Estimate the *probable resource shortage* to avoid disruptions induced by evolution of used exogeneous services.
- Estimate the *probable cost* induced by the resource shortage.
- Identify the services that will have the highest probable cost.
- Estimate if the probable resource shortage is *situational* or *structural*.

To calculate these indicators, we will first list the data necessary for the model to work. Except for the evolution probability distribution that must be provided for each service, the model uses data that is reasonably available within the environment of an organization. We will then detail the set-up of the model and present some assumptions.

8.1 Model data

The data elements that are used in the model are the following:

- The set of used exogeneous services, including for each one:
  - the start time;
  - the termination period;
  - the estimated time to adapt the applications using the service from the current version to a new version;
  - the evolution probability distribution.
- The set of provided services, including for each one:
  - the daily unavailability cost.
- The set of internal applications, including for each one:
  - the daily unavailability cost.
- The organization's dependency graph.

---

81 Because of a rare conjunction of events.
8.2 Set-up, assumptions and hypotheses

Let us consider an organization providing $P$ services that depend themselves, through the organization's application dependency chain, on $C$ consumed services offered by different service providers as shown in Figure 117.

![Figure 117. Setup](image)

8.2.1 Adequate temporal condition

In the model, we suppose that any offered service by an organization has adequate\(^{82}\) temporal conditions. This must be also be true for all consumed services that are considered not to be commodity services, i.e. services that are not provided by more than one provider. In case a service\(^{83}\) is offered by several service providers, it is not necessary for an offered service to have adequate temporal conditions. The consumer of such service can always switch to another similar service offered by another provider in case a service he consumes is phased out.

8.2.2 Termination warning and new service version

In our model, we also make the hypothesis that a service termination warning is sent simultaneously with the new service specification. We also suppose that the new service is available at the time the new service specification\(^{84}\) is sent.

8.2.3 Trained resources and immediate availability

We also suppose that if a service consumer has received one or more termination warnings, he immediately starts the adaptation work of his local applications using services that have changed. This means that the organization never delays the start of the adaptation work if there is some to be done\(^{85}\).

\(^{82}\) As defined in section 7.3.

\(^{83}\) Or any other service providing the same functionality.

\(^{84}\) At the time of the termination warning of the current service.

\(^{85}\) No lazy behavior.
8.2.4 Service evolution randomness and independence of evolution events
We also suppose that any service evolution event is a random event and that all evolution events are independent. This means that we suppose that there are no clusters of evolution events.86

8.2.5 Time to adapt – worst case assumption
We assume that the nature of the future evolutions of a service is not known. Therefore, it is usually not possible to precisely estimate the time that will be required to perform these future adaptations for a service. We also assume that, in most cases, adapting the local integration of a service in an application should take less time and resources than the initial development using the service for the first time. Therefore, in the case no information about a future adaptation exists, a safe estimation for the time needed to adapt the service's local integration code in an application is the time that was needed to develop the initial local integration code for that service.87 Although this seems to be a limiting assumption for the model's usage in real situations, it is not significant in the development of the model. We will show in chapter 11 how this assumption can be easily relaxed for the effective use of the model. This assumption is only made to simplify the model's development.

8.3 Blueprint
The way the model works is the following:

- Definition of a time range from the present up to some time in the future.
- Segmentation of the time range in "time slices" as shown in Figure 118.
- Computation of some indicators for each "time slice".
- Aggregation of the time-based computations results over time.

To ease the model's understanding, we will proceed by steps. We will first start with a basic model for which the time range is limited to a single time slice, the present. We will then define the basic indicators and explain how to use the model on the "time slices" of the future. We will further transform the model to take into account the costs. We will finally show how to use the model with examples.

---

86 This is not true in reality, since often a service provider will often evolve a set of services at the same time, in particular when the services are part of a complex cross-organizational conversation as described in section 4.4. Nevertheless, we will consider the service evolutions as being all independent in our model.

87 Worst case boundary assumption.
8.4 Basic model

For the sake of simplicity of the first explanation, let's assume that all services have the same unavailability cost. We will later add costs in the cost-based version of the model. We will also consider only the present time. We will not worry about the future "time slices" as explained in the model's overview in section 8.3.

Further, we will also suppose that there is no ongoing work stack. This means that we suppose that the human resources that are dedicated to the maintenance and to the adaptation of the evolving services have no current work that could prevent them to immediately start adapting the applications using evolving services.

8.4.1 Task order

Since the unavailability costs are identical for all services in this simplified version, we suppose that maintenance tasks will be performed in increasing order of the contractual deadlines. This is due to the fact that following this strategy will always yield the lowest costs. Please refer to [146] for the details and proof of this. This means that the order of maintenance tasks will resemble the one shown in Figure 119.

88 The ones that receive a termination warning.
8.5 Indicators

For all following calculations, we suppose that all termination warnings in a set of services to adapt are sent at the same time. We have shown elsewhere [147] that it is the "worst case" in terms of ability of handling the case and that any other case in which the termination warnings are not perfectly synchronized can only be a more favorable case. We also consider that there is no current workload for the service integration team at the time of the computation.

The calculated probabilities are the probabilities involving simultaneous evolution events at the time of the calculation. They do not include the consumed services that have already received an evolution event and are already stacked in the amount of adaptation work to be performed by the organization. It is possible to take into account the ongoing work stack by considering the already scheduled adaptation jobs as evolution events with a 100% occurrence probability and by adapting their termination periods and their adaptation times.
8.5.1 PERC-SUCCESS

The PERC-SUCCESS indicator is represented as a $2 \times N$ array of numbers with $N$ being the number of exogeneous services that are used by the organization as shown in Figure 120. The first line contains the number $k$ of simultaneous evolution events. The second line represents the percentage of combinations of $k$ events that can be successfully handled by the organization in the contractual time frame.

<table>
<thead>
<tr>
<th>number of simultaneous evolution events</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5 to N</th>
</tr>
</thead>
<tbody>
<tr>
<td>percentage of cases that can be handled</td>
<td>100%</td>
<td>93%</td>
<td>54%</td>
<td>8%</td>
<td>0%</td>
</tr>
</tbody>
</table>

Figure 120. PERC-SUCCESS indicator example

It is to note that these values do not represent the probabilities for an organization to be able to handle successfully $k$ simultaneous events, but only the percentage of cases that could be theoretically handled among all cases with $k$ simultaneous events. It is based only on the time to adapt the services. The steps for calculating the PERC-SUCCESS indicator are the following:

1. We define $S=\{S_1, S_2, ..., S_k, ..., S_N\}$ as the set of the $N$ consumed services.

2. For each possible subset $R$ of $S$ with $M$ elements ($1 \leq M \leq N$), we create an ordered set of services containing the elements of $R$ denoted $OR = \{OR_1, OR_2, ..., OR_k, ..., OR_M\}$. The elements of $OR$ are ordered according to their termination periods as shown in Figure 121.

3. We define $TP_{OR}=\{TP_{OR,1}, TP_{OR,2}, ..., TP_{OR,k}, ..., TP_{OR,M}\}$ as the set containing the termination periods corresponding to the services of the

89 Which means that the corresponding applications using that service have been adapted in time.

90 The service with the shortest termination period (nearest deadline) is first, the service with the longest termination period is last. Recall that the termination period is defined by contract for each service.
ordered set OR.

4. We define \( \text{ET}_{\text{OR}} = \{ \text{ET}_{\text{OR,1}}, \text{ET}_{\text{OR,2}}, \ldots, \text{ET}_{\text{OR,k}}, \ldots, \text{ET}_{\text{OR,M}} \} \) the set of the estimated times to adapt the corresponding services of OR as shown in Figure 122.

![Figure 122. TP\textsubscript{OR}, ET\textsubscript{OR}](image)

5. Now, based on the hypothesis that the adaptation with the shortest deadline is always performed first\(^1\), we define \( \text{ET}_{\text{COR}} = \{ \text{ET}_{\text{COR,1}}, \text{ET}_{\text{COR,2}}, \ldots, \text{ET}_{\text{COR,k}}, \ldots, \text{ET}_{\text{COR,M}} \} \) the set of cumulative adaptation times due to evolution events, where \( \text{ET}_{\text{COR,i}} \) is defined as:

- \( \text{ET}_{\text{COR,i}} = \text{ET}_{\text{OR,i}}, \) for \( i = 1 \)
- \( \text{ET}_{\text{COR,i}} = \text{ET}_{\text{COR,i-1}} + \text{ET}_{\text{OR,i}}, \) for \( 2 \leq i \leq M \)^2

6. We now compare the elements of the cumulative ordered estimation evolution times set \( \text{ET}_{\text{COR}} \) to the corresponding elements of the ordered termination periods set \( \text{TP}_{\text{OR}} \) to determine if the combination of termination warnings could be successfully handled in time by the organization. This can be expressed in pseudo-code as:

\[
\text{if } (\forall i \in [1,M], \text{ET}_{\text{COR,i}} \leq \text{TP}_{\text{OR,i}}) \text{ then } \text{SUCCESS}_k = \text{SUCCESS}_k + 1
\]

where \( \text{SUCCESS}_k \) counts the number of combinations of \( k \) termination warnings that can be successfully handled.

7. These steps are repeated for each subset \( R \) of \( S \) with \( R \) having \( M \) elements, with \( 1 \leq M \leq N \) and where \( N \) is the number of consumed services of the organization.

8. We now define \( \text{SUCCESS} = \{ \text{SUCCESS}_1, \ldots, \text{SUCCESS}_k, \ldots, \text{SUCCESS}_N \} \)

---

\(^{1}\) As shown in [147].

\(^{2}\) Because the adapting of service \( i \) would only start when service \( i-1 \) has been adapted.
as the set containing all SUCCESS\_k elements ordered after k.

9. We now compute \( \text{TOTAL} = \{ \text{TOTAL}_1, \ldots, \text{TOTAL}_k, \ldots, \text{TOTAL}_N \} \) as the ordered set of the number of combinations of k events, with \( 1 \leq k \leq N \). Each element is given by:

\[
\text{TOTAL}_k = \frac{N!}{(N-k)! \cdot k!}
\]

10. We finally calculate PERC-SUCCESS as the ordered set containing for each k the percentage of cases of k simultaneous evolution events that can be handled successfully. Every element of PERC-SUCCESS is defined as:

\[
\text{PERC-SUCCESS}_k = \frac{\text{SUCCESS}_k}{\text{TOTAL}_k}
\]

### 8.5.2 PROB

The PROB indicator consists of a \( 2 \times (N+1) \) array of numbers, with N being the number of exogeneous services used by the organization. The first line contains the number k of simultaneous evolution events. The second line represents the probabilities of receiving k simultaneous evolution events. These probabilities are calculated based on the available information of the expected evolution of the consumed services as discussed in section 7.5.

<table>
<thead>
<tr>
<th>number of k simultaneous evolution events</th>
<th>0</th>
<th>1</th>
<th>...</th>
<th>N-1</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>probability of having exactly k simultaneous events</td>
<td>81%</td>
<td>15%</td>
<td>...</td>
<td>&lt;10^{-15}</td>
<td>&lt;10^{-17}</td>
</tr>
</tbody>
</table>

*Figure 123. PROB indicator table*

Since each service consumed by the organization has its own evolution probability distribution, its probability at the time of the computation is also unique. Thus, there is no ready-to-use formula or method\(^93\) to calculate the probability of receiving a given number of termination warnings. To perform this computation, it is necessary to define the complete set of all possible outcomes, to calculate the probability for each outcome then to sum up the probabilities of each group of outcomes that possess the same number of issued termination warnings.

However, there are only two possible outcomes when considering a termination

---

\(^93\) Because of the different probabilities of these different events, there is no method similar to Bernoulli trials nor any generalized method using the inclusion-exclusion principle applied to probability.
warning event: it happens or it does not happen. The probability that the termination warning occurs at the day of the computation is the probability slice for that day. We define the outcome as an occurrence when the termination warning occurs. The probability of an occurrence for service \( S_k \) is noted \( p_k \), with \((1 \leq k \leq N)\). We define the outcome as a non-occurrence when the termination warning does not occur. The probability of a non-occurrence for service \( S_k \) is noted \( q_k = 1 - p_k \). To determine the probabilities of several simultaneous occurrences, it is necessary to define the N-tuple space containing vectors of all individual outcomes, with \( N \) the number of services consumed by the organization which could receive a termination warning. The complete set of outcomes in the N-tuple space is the combination of all failures and successes for all consumed services in the organization as shown in Figure 124. Therefore, there are \( 2^N \) possible outcomes in the N-tuple space.

<table>
<thead>
<tr>
<th></th>
<th>Service 1</th>
<th>Service 2</th>
<th>...</th>
<th>Service N-1</th>
<th>Service N</th>
</tr>
</thead>
<tbody>
<tr>
<td>case 1</td>
<td>non-occurrence</td>
<td>non-occurrence</td>
<td>...</td>
<td>non-occurrence</td>
<td>non-occurrence</td>
</tr>
<tr>
<td>case 2</td>
<td>non-occurrence</td>
<td>non-occurrence</td>
<td>...</td>
<td>non-occurrence</td>
<td>occurrence</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>case ( 2^{N-1} )</td>
<td>occurrence</td>
<td>occurrence</td>
<td>...</td>
<td>occurrence</td>
<td>non-occurrence</td>
</tr>
<tr>
<td>case ( 2^N )</td>
<td>occurrence</td>
<td>occurrence</td>
<td>...</td>
<td>occurrence</td>
<td>occurrence</td>
</tr>
</tbody>
</table>

Figure 124. Set of possible outcomes in the N-tuple space

### 8.5.2.1 Computation

The steps for calculating the PROB indicator table are the following:

1. For each possible subset (case) of services \( R \) that could receive termination warnings, we calculate the probability for that subset to happen. Since all events are independent, this calculation is done by multiplying the probabilities of the individual outcomes of each termination warning event:

\[
P(\text{case 1}) = q_1 \cdot q_2 \cdot ... \cdot q_{N-1} \cdot q_N
\]

\[
P(\text{case 2}) = q_1 \cdot q_2 \cdot ... \cdot q_{N-1} \cdot p_N
\]

\[
...\]

\[
P(\text{case } 2^{N-1}) = q_1 \cdot q_2 \cdot ... \cdot p_{N-1} \cdot p_N
\]

\[
P(\text{case } 2^N) = p_1 \cdot p_2 \cdot ... \cdot p_{N-1} \cdot p_N
\]

2. We define an array \( P \) of \( N \) elements. The value of each element of the array is initialized to 0.

94 By hypothesis (see section "Setup and assumptions")
3. We then iterate through the set of calculated probabilities. For each calculated probability, we count the occurrences and denote that number \( k \)occurrence. We add the calculated probability to the value of the \( k \)occurrence -th element of the array \( P \).

### 8.5.2.2 Calculation optimizations

Due to the exponential nature of the calculation, the number of computations increases rapidly. To reduce the calculation time, it is necessary to limit the number of computations by performing only the meaningful ones. This can be done by ordering the computation in a way that is slightly different than the one shown in Figure 124 and by defining a probability threshold. This probability threshold is used as a stop condition in the computation algorithm. As the probability for each case is calculated, the cumulative sum of all calculated probabilities tends to 100%. Depending on the \( p_k \)'s, it is appropriate to order the computations relative to \( k \). This means that we first calculate the probabilities of all the cases for a certain \( k \), then for another \( k \), etc. After performing the calculations for all cases of a certain \( k \), the algorithm checks if the remaining probability is smaller than the threshold and exits if it is the case.

If a majority of the \( p_k \)'s are well under 0.5, it is less expensive to compute the results following the path in an ascendant way (with \( k = 0 \) to \( N \)) as shown in Figure 125. This means that the algorithm will perform first the computation for the case of 0 occurrences, then for all cases of 1 occurrence, then for all cases of 2 occurrences, etc., up to the case of \( N \) occurrences. After calculating the probabilities for all cases of \( k \) occurrences up to a certain \( k \), the remaining computations are not performed because all further results will only be smaller than the threshold and can therefore be considered as being of small significance.

![Figure 125. Sum of probabilities for all cases of \( k \) with majority of \( p_k \) \( << \) 0.5](image)

Similarly, if a majority of the \( p_k \)'s is well over 0.5, it is less expensive to compute the results following the path in a descendant way (with \( k = N \) to 0).
8.5.3 PROB-SUCCESS

The PROB-SUCCESS indicator consists of a $2 \times (N+1)$ array of numbers, with $N$ being the number of exogeneous services used by the organization as shown in Figure 126. The first line contains the number of simultaneous evolution events. The second line represents the probabilities of receiving and handling successfully the $k$ simultaneous evolution events.

<table>
<thead>
<tr>
<th>number of $k$ simultaneous evolution events</th>
<th>0</th>
<th>1</th>
<th>...</th>
<th>N-1</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>probability of receiving and handling successfully $k$ simultaneous events</td>
<td>81%</td>
<td>12%</td>
<td>...</td>
<td>~0%</td>
<td>~0%</td>
</tr>
</tbody>
</table>

Figure 126. PROB-SUCCESS indicator table

The computation of this indicator is similar to the one of the PROB indicator. The main difference is that for each calculated probability of each combination of termination warnings, the success of the handling of that combination is calculated as in the PERC-SUCCESS indicator.

8.5.3.1 Computation

The steps for calculating the PROB-SUCCESS indicator table are the following:

1. For each possible combination (subset) $R$ of services that can receive termination warnings, we calculate the probability that it could happen (idem as step 1 of the PROB indicator) and compute the possibility of handling it successfully (idem as in steps 2, 3, 4 and 5 of the PERC-SUCCESS indicator).

2. If this combination of termination events can be handled successfully by the organization as determined in step 2, we add the probability to the value of the $k^{th}$ corresponding cell of the PROB-SUCCESS array, with $k$ being the size of $R$.

Reminder: the sum of all the probabilities for all $k$ is less or equal to 100% since the probability for each $k$ is less or equal than the $k^{th}$ corresponding probability in the PROB indicator.

---

95 This means that all adaptations can be done within the organization's contractual time frames of the consumed services.
8.5.4 EVOL-EXP

The EVOL-EXP indicator consists of the probability for an organization of being able to handle successfully the evolution events that could occur at the time of the computation. This probability takes all possible simultaneous evolution events into account. We denote this probability as the *evolution exposure* which can be seen as the synthetic answer to the question:

"What is the probability that an organization will be able to adapt to the evolutions of its outside world within a predefined time frame such that it will not suffer any disruption due to these evolutions?"

The *evolution exposure* indicator represents the probability of receiving termination warnings and being able to perform the adaptations successfully within the contractual time frame of an organization. The EVOL-EXP indicator is calculated by summing up all probabilities of the PROB-SUCCESS indicator:

\[
\text{EVOL-EXP} = \sum_{i=1}^{N} \text{PROB-SUCCESS}[k]
\]
8.6 Adding work stack

Until now, for the sake of simplicity, we made the assumption that there is no work stack at the time of the computation\textsuperscript{96} for the indicators. Since this is not the case in a real-world situation, we must take into account the ongoing adaptation tasks that are the consequence of termination warnings that already happened before the present time of the computation.

First, we must know what the current work stack is. This means that there is a certain number $M$ of tasks that we add to the stack of the services receiving a possible termination warning. At the time of the computation, we know \textit{with certainty} that the termination warnings for these past tasks have \textit{already been issued}. Thus we can add artificially $M$ “new services” to the list of services that may receive a termination warning. Each of this “new services” will have the following deadline:

$$\text{deadline} = \text{day of the termination warning} + \text{duration of contractual termination period}$$

The duration of the adaptation time is the remaining duration to perform the changes at the time the calculation is done. The probability of the “new services” to get a termination warning at the time of calculation is 1 as shown in Figure 127.

![Figure 127. Services eligible for termination with current work stack](image)

By doing so, we can take into account all service versions that have already been...

\textsuperscript{96} For the moment, the present.
notified for termination and that have not yet been adapted. Thus, if we consider the previous example shown in Figure 127, the computation of the indicators supposing the occurrence of a termination warning of Service 1 and Service 3 would lead to the task order shown in Figure 128.

![Figure 128. Services eligible for termination with current work stack](image)

As seen above, the jobs of the work stack can simply be considered as services with a probability of 1 of receiving an evolution event at that instant.

### 8.7 At present, now in the future

Now that we can calculate these indicators at time \( t = \text{present} \), we would like to extend this by considering not only the present but also the future. We would like to compute the indicators for the time slices of the future as shown in Figure 118 on page 143 the same way we did for the time slice of the present in the precedent sections.

#### 8.7.1 Problems to solve

There are 2 major problems to solve when one wants to apply the basic model to the time slices in the future:

- The exponential number of computations
- The estimation of the work stack.

#### 8.7.1.1 Exponential number of computations

The first problem is that the number of computations needed to apply the model to
time slices in the future grows exponentially in the case we would like to compute the indicators with the same precision as we did for the present. Thus, it is necessary to find a way to reduce the number of cases to be computed to the most relevant ones.

Hence, for the problem of number of computations, we will consider the following approaches:

- Calculating the indicators for all possible futures in section 8.7.2.
- Calculating the indicators for the most probable futures of each time slice in section 8.7.3.

8.7.1.2 Estimation of the work stack

The second problem is that our model needs to have the work stack of the time slice for which the indicators are computed to work properly. But at this stage, we don't know what the work stack will be for the time slices in the future. We only know the work stack at the present. In fact, there is no single work stack for a time slice at a time \( t \) in the future representing well the effective future work stack at time \( t \) because there are too many situations that could exist in the future at time \( t \). Estimating one single work stack would be as if we were knowing the future itself at time \( t \) or as if we were implying that the future at time \( t \) has very little possible volatility. Therefore, we will not estimate one single work stack for each time slice of the future. Instead, we will present a method in which we will estimate multiple work stacks for each time slice of the future.

More precisely, based on the fact that each service version can receive a termination warning only once, we will show in sections 8.7.4.1 and 8.7.4.2 how each combination of evolution events that could take place at time \( t \) constrains the possible past states at time \( t \) if that combination would effectively take place at time \( t \). Thus, we will estimate multiple work stacks for each time slice \( t \) in the future.

We will show that instead of performing \( 2^N \) computations for each time slice as done in section 8.5.2, we must perform \( L \cdot \sum_{k=0}^{N} \left( \binom{N}{k} \cdot \binom{N}{N-k} \right) \) computations, with \( N \) the number of exogeneous services and \( L \) an integer influencing the precision of the computations.\(^99\).

---

97 The one that will really happen.
98 It is as if we would make the assumption that all possible states at time \( t \) will be very close to each other, which is completely unknown.
99 In reality, \( L \) is a number of simulations to be performed for each case. The higher the number of simulations, the higher the precision. Hence the term "influencing the precision".
8.7.2 Calculating the indicators for all possible futures

**Definition – A future**

We define a future $F_t$ as one of the states that may exist at the time $t$ to come. This does not designate the effective state that will be at time $t$, but only one of the possible states that could exist at that time.

If we would like to calculate the indicators exhaustively for all possible futures, we would need to consider every combination of termination warnings for all services over all future time slices. For this, we would first determine for each combination of termination warnings the most rational order in which the adaptations should be done, and then compute the probability weighted indicators which would yield values for the indicators over time. However, this is not possible. Since we are calculating all possible futures over time, the number of cases grows much too quickly to be tractable.

Let there be $N$ independent services and let us set time $t=t_0$ to indicate the present time. We consider a time period from $t_0$ to $t_{T100}$. At each time $t^k_{101}$ with $0 \leq k < T$, each service $S_i$, $1 \leq i \leq N$, has a probability $p_{i,t_k}$ of receiving a termination event or a probability $q_{i,t_k}$ of not receiving it. This means that there is a number of possible situations equal to $2^N$ at time $t_k$. If the termination events at time $t_{k+1}$ would be independent from the ones at time $t_k$ there would be again $2^N$ possible states at time $t_{k+1}$. This would lead to a number of possible cases equal to $(2^N)^T$.

But a service version can receive a termination warning at most once during its lifetime, and therefore also only once within the time interval $[t_0, t_{T}]$. Thus, each service version has $T$ possibilities of receiving a termination warning during $[t_0, t_{T}]$ or it could not receive any termination warning at all. Hence, each service version will have $T+1$ states during the time interval $[t_0, t_{T}]$ in terms of evolution events. Extending this to all services, this means that there can be $(T+1)^N$ states in terms of evolution events for the set of $N$ services. As an example, it means that for a setup with 10 services and an evolution probability distribution covering a range of 150 days, there would be around $151^{10} = 5.77 \times 10^{21}$ cases to calculate. If we would calculate the probabilities of all possible futures, the sum of all probabilities would sum up to 1.
8.7.3 Calculating the indicators for the most probable futures of each time slice

Since computing all possible futures induces too many computations, we propose to compute only certain futures for each time slice. The idea is that since a service version receives only once a termination warning, every combination of termination warnings at time t will constrain its past. Thus, the number of possible past states that could exist in the past for a certain combination of evolution events at time t will be only a subset of all possible states that could exist in the time interval \([t_0, t]\). Moreover, we will choose for each combination of evolution events at time t only the most probable states that would exist in the time interval \([t_0, t]\) if that combination takes place at time t. For each of these states, we will calculate the indicators and weight them with the probability of the state to exist.

**Definition – Past state of the future F**

We define a past state of a future \(F_t\) as any state in the past of a future \(F\) at time t.

Knowing that for each service a termination warning is issued only once, then for each combination of termination warnings \(C_{tw,t}\) at time t, we can constrain the number of possible past states of all the futures \(F_t\) that could exist if that combination of termination warnings takes place at time t.

This number of possible past states is the number of combinations of all issuances or non-issuances of termination warnings that have not taken place in the combination \(C_{tw,t}\). This does not mean that we know at time \(t_0\) what past state of the future \(F_t\) will be effectively the realized past state of the future \(F_t\). We only know the probability of each possible past state of \(F_t\) conditioned by the combination \(C_{tw,t}\). This probability is defined as followed.

**Definition – Past state probability**

A past state probability is defined as the probability of a past state of the combination \(C_{tw,t}\) of a futures \(F_t\) to be the effective past state of \(F_t\).

Again, the number of past states of a future \(F_t\) conditioned by the combination \(C_{tw,t}\) can still be too high such that all computations can be performed exhaustively. This number would be the sum of all \(C_{tw,t}\) combinations for each \(C_{tw,t}\) combination which amounts to \(\sum_{k=0}^{N} (C_k^N (t+1)^{N-k})\) computations for \(t=T\). This is because it would include all combinations of all termination warnings not part of
the combination $C_{t_0,t}$ for all times\textsuperscript{102} in the interval $[t_0, t]$. Thus, we choose to restrict the computation to the most probable past state of each combination $CPS_{t_0,t}$ of past state termination warnings not part of $C_{t_0,t}$. By doing this, we reduce the number of computations to $\sum_{k=0}^{N} (C_k^N \cdot C_{N-k}^N)$. Further, like in section 8.5.2.2, we can limit the computation time by setting a probability threshold that in effect stops the computations as soon as the threshold is reached.

But the time at which each termination warning part of the combination $CPS_{t_0,t}$ has been issued for the most probable combination $CPS_{t_0,t}$ is not known at this stage. If we can then estimate the time of each realized termination warning part of the combination $CPS_{t_0,t}$, we can then estimate a probable future $F_t$. This means that we have then $2^M$ estimated futures $F_t$ for each combination $C_{t_0,t}$, with $M$ being the number of termination warnings not part of $C_{t_0,t}$.

### 8.7.3.1 Total number of computations over the foreseeable future

To calculate the total number of computations needed by the model at this stage, we will now go again through the evolution exposure indicator described in section 8.5.4. The evolution exposure calculated for the time $t$ is dependent on the work stack as defined in section 8.6. This if fine if time $t$ is the present, because the work stack is known with certainty at the present. But if we want to calculate the evolution exposure for a time $t$ in the future, we need to know the work stack that exists at that time. Since $t$ is in the future, we don't know it. But the computation of the evolution exposure is an aggregate of probability-weighted computations each assuming one of the possible conjunctions of termination events that can happen at time $t$.

Since we don't know the work stack at time $t$, we cannot simply perform the $2^N$ computations to calculate the evolution exposure at time $t$ as described in section 8.5.4. Instead, we can calculate the $\sum_{k=0}^{N} (C_k^N \cdot C_{N-k}^N)$ probability-weighted computations. Each of these $\sum_{k=0}^{N} (C_k^N \cdot C_{N-k}^N)$ probability-weighted computations needs the work stack to be defined. Therefore, we will estimate the work stack of each computation. To see the evolution through time of the indicator, we must do this for each time slice in the foreseeable future from $t_0$ to $t_T$. Hence, there will be a total number of $T \sum_{k=0}^{N} (C_k^N \cdot C_{N-k}^N)$ computations.

\textsuperscript{102} Discreet times.
8.7.4 Estimating the work stack

The estimation of the work stack for each probable future $F_t$ conditioned by a combination $C_{tw,t}$ can be done in two different ways. The first one is a stable one and fast to compute, but it can introduce a bias on estimated times of termination warnings. The second one is not stable, because it is based on multiple random simulations following an implied probability distribution, and it is slower to compute. Both methods share the same foundations, the second one being a refinement of the first.

8.7.4.1 Estimation of the work stack for each probable past of a future $F_t$ conditioned by a combination $C_{tw,t}$

To illustrate how we calculate an indicator at a certain time $t$ in the future and if this indicator is a probability-weighted aggregate of computations implying each a certain conjunction of termination events at time $t$, we will use the probable resource shortage indicator as an example. We will not detail the computation of the indicator but use it only as an illustration for the estimation of the work stack.

8.7.4.2 Probable resource shortage indicator

Definition – Probable resource shortage

The probable resource shortage at time $t$ is defined as the difference between:

- The sum of all man-days that an organization would have needed to perform all necessary adaptations such that it would not suffer any disruptions due to a conjunction of termination events at time $t$. This sum is calculated using a closest-deadline first ordering of tasks as described in section 8.4.1.

and

- The sum of all available man-days for performing these adaptation tasks at time $t$ when using a closest-deadline first ordering of tasks as described in section 8.4.1.

To calculate the probable resource shortage indicator with a defined work stack at time $t$, we first calculate the excess number of days needed to adapt the

---

103 In the sense that if the computation is done again, it will produce exactly the same result.
104 In the sense that if the computation is done again, it will produce a close result.
applications for each combination $C_i$ of termination warnings, with $1 \leq i \leq 2^N$ and $N$ being the number of used exogeneous services. We denote each of these values $\text{Shortage}_i$. We then sum up each of these $\text{Shortage}_i$ multiplied by the probability $p_i$ of the corresponding combination of termination events to happen,

$$\text{Sum} = \sum_{i=1}^{2^N} (\text{Shortage}_i; p_i),$$

which is the estimated resource shortage value at time $t$. This amounts to performing $2^N$ computations.

But since the $\text{Shortage}_i$ depend on the work stack and since the work stack in the future is not defined, we cannot know the $\text{Shortage}_i$. But for each combination $i$ of termination events at time $t$, we can calculate each past state’s probability $\text{Pps}_{i,j}$, $1 \leq j \leq 2^M$, $M$ being the number of termination events not occurring in $C_i$ at time $t$. This means that we must not do only $2^N$ computations, but we must do $\sum_{k=0}^{N} \binom{N}{k} \cdot \binom{N}{N-k}$ computations, $N$ being the number of services.

We define $\text{NOC}_{i,j,t}$ as the $j$-th combination of the non-occurring termination events of the combination $i$ at time $t$. There is a number $2^M$ of $\text{NOC}_{i,j,t}$. Thus, $\text{Pps}_{i,j}$ is the probability that $\text{NOC}_{i,j,t}$ happens and is computed as follows.

Each $\text{Pps}_{i,j}$ is calculated similarly as described in section 8.5.2, only with different probability values. Thus, the probability of a termination event happening between the present and time $t^{105}$ is the cumulative daily probability of the service evolution probability distribution between the present and time $t^{106}$.

We define $\text{Pcumulative}_{i,j,t,r,k}$ the cumulative probability of the occurrence of the $k^{th}$ termination warning part of $\text{NOC}_{i,j,t}$ between the present and time $t$. Similarly, we define $\text{Qcumulative}_{i,j,t,r,k}=1-\text{Pcumulative}_{i,j,t,r,k}$

We can now express $\text{Pps}_{i,j}=\text{Pcumulative}_{i,j,t,r} \cdot \text{Qcumulative}_{i,j,t,s}$ with $r$ being the $r^{th}$ happening termination event of $\text{NOC}_{i,j,t}$ and $s$ being the $s^{th}$ non-happening termination event of $\text{NOC}_{i,j,t}$.

We now define $\text{EstimatedShortage}_{i,j}$ as the excess number of days needed to adapt the applications when considering a particular $\text{NOC}_{i,j,t}$.

Finally, we can now restate the estimated resource shortage value at time $t$ from

$$\text{Sum} = \sum_{i=1}^{2^N} (\text{Shortage}_i; p_i)$$

---

105 Time $t$ excluded.
106 Idem.
Managing evolution risk of cross-organizational services

\[
\text{Sum} = \sum_{i=1}^{\infty} \left( \sum_{j=1}^{M_i} Pps_{i,j} \cdot \text{EstimatedShortage}_{i,j} \cdot p_i \right)
\]

It is important to remember that at this stage we do not know yet the value of \(\text{EstimatedShortage}_{i,j}\), but we already know how to calculate \(Pps_{i,j}\). Thus, the only necessary thing that is left to do is to estimate the time at which the termination events of \(NOC_{i,j,t}\) take place, such that we can estimate a work stack at time \(t\), and consequently estimate the \(\text{EstimatedShortage}_{i,j}\) values.

This is done by taking the time corresponding to the median of the probability that would have become certainty between the present and time \(t\) as shown in Figure 129.

The problem with this approach is that if we have many services that have low probability for a certain amount of time, we introduce a bias for the consequences of conjoint termination warnings because the estimation of the time termination warnings will be very similar until the probability curve differs as shown in Figure 130 even in case of very different distribution shapes afterwards.

The consequence of having the same estimated evolution times for all \(NOC_{i,j,t}\) is that it inflates possible shortage of resources because since all termination events are estimated to take place at the same time, it increases the joint probability of trespassing the deadlines.
To limit this effect, we have developed an extended approach to perform the computations. We first derive the implied probability distribution between the present and time $t$ as a new complete probability distribution labeled $P_{\text{implied}}$ as shown in Figure 131.

We then perform a number $L$ of iterations in which we determine the estimated time of the termination warnings of $\text{NOC}_{i,j,t}$ randomly. These random values follow the implied probability distribution $P_{\text{implied}}$. This is done by using a number generator using a uniform distribution and mapping the uniformly distributed random value to $P_{\text{d}}$ as shown in Figure 132. In short, we perform $L$ random
tests that use the probability distribution up to time $t$.

![Figure 132. Uniform to implied distribution mapping](image)

We then determine the estimated time $t$ from the inverse of the value of $P_{implied}$ as shown in Figure 133. We then compute $L$ times the indicator for each $\text{NOC}_{i,j,t}$ and average the aggregated result of these calculations.

![Figure 133. $P_{implied}$ randomly estimated evolution times](image)
The advantage of this method is that we get rid of the estimated time of termination warning bias as shown in Figure 133 and therefore calculate much more precisely the value of the probable resource shortage indicator. When calculating the costs indicator that will be described in section 8.8, we have made numerous tests which have shown that this method tends to better estimate the costs than the previously described biased one, which tends to overestimate costs because of the non-linear relationship of the computation. The main disadvantage is that the computation is L times longer than the biased method. This means that when using this method, there are not $\sum_{k=0}^{N} C_N^k \cdot C_N^{N-k}$ computations, but $\sum_{k=0}^{N} \left( C_N^k \cdot C_N^{N-k} \cdot L \right)$ computations. It is important to remember that for each of these computations, we must perform an ordering of the tasks based on the deadline-first approach described in section 8.4.1.

8.8 Adding costs

Until now, our approach has not differentiated the "value" of consumed and offered services. It has only focused on the probabilities of successfully handling a future situation. This approach takes into account the estimated probabilities of each of the consumed services to be subject to a future evolution event, and takes also into account all the combinations of all these events. Depending on the ability of handling a combination of particular events, we depict the exposure of an organization to the evolution of its outside world through different indicators.

One of the strong assumptions we use through all the calculations is that the different adaptation tasks corresponding to evolution events are ordered by performing the tasks with the shortest deadlines first. This approach permits to guarantee that if there exists at least one order allowing to handle successfully within the contractual time frame a particular combination of evolution events, the combination order with the shortest deadlines first will find with certainty one solution satisfying all deadline constraints. It means that if the combination order with the shortest deadlines first does not yield a solution, i.e. that at least one deadline is not respected, then there is no solution whatever the order [146].

This first approach so far does not take into account the costs resulting from the unavailability of a particular consumed service. These costs can be either of internal or external nature. The internal costs comprise all the costs that can be associated with tasks performed by internal persons of an organization that become unavailable when a consumed service becomes unavailable. As shown in Figure 134, this includes all tasks that are dependent on the consumed services.
The external costs comprise all the costs that are the consequence of the unavailability of offered services. In our context, we narrow the definition to the costs linked to the unavailability of offered services due to the unavailability of consumed services because they could not be adapted within the contractual timeframe.

### 8.8.1 Cost-based task order

Considering costs in such a system changes strongly the goals that an organization's manager would have in terms of minimization. In that context, the goal is not to minimize the probability of being unable to handle the evolution events within the contractual time frame, but it is to minimize the probable costs that will arise from the whole system due to the impossibility of performing some internal tasks and also due to the unavailability of services offered to other organizations. These unavailable offered services can't be operated and therefore can't be billed. This can even generate unavailability penalties in case of clauses ruling that event in a contractual agreement. Again, we consider only the costs induced by the non-adaptation of local systems after evolution event occurrences. We do not consider any other type of costs like service failure, etc.

In the real world's cost minimization context, the task ordering assumption that we used until now does not hold any more. Thus, in case of a termination event, the task ordering will not be done by considering the task's deadlines, but will be done based on the costs that would occur if the task is done before or after some other tasks. This cost is the total cost of all services during their respective unavailability periods.

For example, let's consider the set of tasks shown in Figure 135 with their associated costs in case of unavailability.
If termination warnings occur simultaneously for all these services, the ordering of the adaptation tasks using the termination deadlines would be as shown in Figure 136.

But ordering the tasks relatively to the minimum costs would not be using the services termination deadlines as shown Figure 137.
This simple example illustrates that the order of tasks that minimize costs is not necessarily the one following the termination deadlines. The problem of ordering these tasks based on tardiness costs is in fact a very well known and researched problem: it is known in the literature as the job-shop scheduling problem.

### 8.9 Job-shop scheduling

The problem we face is the classical single machine total weighted tardiness problem as classified in [148]. This problem is known to be strongly NP-Hard, but there has been a lot of research in this field since the turning of the new millennium ([149], [150]). We used an implementation with publicly available source code solving the more general earliness-tardiness problem ([149], [151], [152]) for cases up to 50 jobs. The single machine total weighted earliness-tardiness problem is described in [151] as:

We consider the problem $1|r_i|\sum \alpha_i E_i + \beta_i T_i$: A set of $n$ jobs $J_1, ..., J_n$ with processing times $p_1, ..., p_n$ has to be scheduled with no preemption on a single machine. For any $i$, $J_i$ is available at time $r_i$ and has a due date $d_i$. If the completion time of $J_i$, denoted by $C_i$, occurs after $d_i$, the job is late and a penalty $\beta_i(C_i - d_i)$ must be paid. Conversely, if $C_i < d_i$, the job $J_i$ is early and the penalty is then $\alpha_i(C_i - d_i)$. The goal is to find a schedule that minimizes the sum of all the penalties.

The source code of the used algorithm being written in C, we have ported it to Java. A more efficient approach described in [150] and claiming to solve cases up to 300 jobs would have probably improved the implementation's speed but at the time of the writing of this document the source code was still not published.
8.9.1 Calculating costs
To be able to calculate the probable costs over time that are induced by the non-adaptation in time of applications to evolving exogeneous services, it is necessary to know for each exogeneous service its the daily cost in case of unavailability. The daily unavailability cost of each service is the sum of all costs that would arise if the service is not available. This comprises the unavailability costs of all dependent applications or provided services to other organizations. Thus, an organization must first estimate what would be the daily unavailability cost for a each of its applications or provided services as shown in Figure 138.

Then, using the organization's dependency graph, we can walk recursively through the dependency tree starting from each exogeneous service. We sum up all costs of all applications dependent of that service.

**Definition – Service unavailability cost**

A service's unit unavailability cost is defined as the total cost that would encur an organization if it is unavailable for one day.
It is important to note that the service unavailability cost of each service is computed regardless of other services. Thus, the daily unavailability costs of different services can each contain the unavailability costs of the same applications and offered services. We will show in section 8.9.4 why this is important and also how it is taken into account in the model.

### 8.9.2 Recomputing all indicators using the cost-based ordering method

All indicators can now be recalculated the same way they have been defined, but instead of using the deadline-first task ordering method as described in section 8.4.1, we use the cost-based ordering method as described in section 8.8.1.

### 8.9.3 Probable costs

If an organization knows its dependency graph and if it has an estimation of the individual daily unavailability cost of each of its applications and provided services, and if it has the service probability distributions for its used exogeneous services, then it is possible to determine the probable cost that the organization will encur over time due to the non-adaptation of these evolving services.

**Definition – Probable non-adaptation evolution costs**

The probable non-adaptation evolution costs at time t of an organization is defined as the total costs that the organization will probably encur starting from time t because it could not adapt its applications to the evolving exogeneous services it is using in the contractual time boundaries it has agreed with its service providers.
The probable non-adaptation evolution costs is referred for the rest of the document as the probable costs.

**8.9.4 Probable costs indicator**

The probable costs indicator is calculated using the same approach as the resource shortage indicator described in section 8.7.4.2. The additional difficulty is that it is necessary to take into account the potential overlap of costs due to the disruption of the same applications and offered services by multiple consumed services as described at the end of section 8.9.1. This is handled as described hereafter.

Since we proceeded like in section 8.7.4.2, at one point we must do each of the \( \sum_{k=0}^{N} (C_k^N \cdot C_{N-k}^N \cdot L) \) computations. For the probable costs indicator, we must calculate the costs of each of the \( \sum_{k=0}^{N} (C_k^N \cdot C_{N-k}^N \cdot L) \) cases.

A naive approach is to compute the tardiness of each service for each of the \( \sum_{k=0}^{N} (C_k^N \cdot C_{N-k}^N \cdot L) \) cases and multiply this tardiness by the daily cost of unavailability of each service. The problem with this approach is that costs of the individual applications and provided services of the organization can be counted several times. Thus, this approach overestimates the "real" probable costs that the organization would incur in that particular situation.

We will use a simple example to explain when this occurs. Let an organization O have one application depending on three exogeneous services S1, S2 and S3 as shown in Figure 140. All three services have an estimated unavailability cost\(^{107}\) of 10. All three services also have an estimated adaptation time of 10 days and a termination period of 15 days. There is no work stack. All three services have evolution probability distributions with a probability of receiving a termination warning at the present that is greater than 0. Let the time t be the present.

---

\(^{107}\) Cost unit per day, whatever it is.
Managing evolution risk of cross-organizational services

At time $t$, one computes the probable costs indicator by performing
\[\sum_{k=0}^{N} (C_k^N \cdot C_{N-k}^N \cdot L)\] computations. In $L$ of them, $S_1$, $S_2$ and $S_3$ will all receive a termination warning. This means that the cost-based ordering will stack the adaptations in an order one after the other. Just for this example, since all services have all the same cost and the same termination period, the determination of the optimal order is irrelevant. Therefore, we arbitrarily set the order of $S_1$, then $S_2$ and finally $S_3$. Thus, the application can be adapted to the changes of service $S_1$ during its termination period, contrary to the changes of $S_2$ and $S_3$ which will therefore be the cause of unavailability of the application as shown in Figure 141.

If the unavailability costs would be calculated without taking the multiple-counting issue into account, the unavailability cost for that $L$ cases would be 160 as shown in Figure 142.
S1 unavailability cost = 0*10 = 0
S2 unavailability cost = 4*10 = 40
S3 unavailability cost = 12*10 = 120

total unavailability cost = 0 + 40 + 120 = 160

Figure 142. Unavailability cost without handling multi-counting of costs

But since the costs originate from the non-availability of the same application, the real costs of each of these L cases is in reality 120. This is the case because the unavailability costs of S2 and S3 are counted twice during their overlapping tardiness as shown in Figure 143.

S1 unavailability cost = 0*10 = 0
S2 unavailability cost = 4*10 = 40
S3 unavailability cost = 8*10 = 80

total unavailability cost = 0 + 40 + 80 = 120

Figure 143. Unavailability cost when handling multi-counting of costs

Thus, we have developed a method that avoids counting the costs twice when the overlapping tardinesses induce costs originating from the same source, whether it is an internal application or a provided service to other organizations. This is done through several steps. First, we create at the very beginning of all the computations a table of cost origin per used exogeneous service. Then we calculate the cost of each of the \( \sum_{k=0}^{N} (C_k \cdot C_{N-k} \cdot L) \) computations in two iterations.

The first iteration is used to record what exogeneous service will suffer tardiness. During that first iteration, we record for each unit of time the services that suffer adaptation tardiness. Then, during the second iteration, we use this table to take into account the tardiness cost only once, and we distribute the shared costs equally between all services that had common cost origins between them for each unit of time. By doing this, we count correctly the costs for each of the
\[ \sum_{k=0}^{N} (C_{k}^{N} \cdot C_{N-k}^{N} \cdot L) \] cases.

It is important to note that this can introduce "shadow costs", because even if we optimize the number of resources allocated to the evolution maintenance tasks like described in section 9.3, or if we renegotiate the termination period of particular services as described in section 9.4, and therefore limit the tardiness of one or multiple exogeneous services, the total cost could diminish less than intuitively expected. The reason for this is because the other services that were suffering tardiness at the same times but for which the costs were not double-counted but equally shared would this time still be counted but shared between less services.

8.10 Model implementation and processing environment

The model and the single machine total weighted earliness-tardiness algorithm described in [151] have been implemented in Java. The model has the following metrics:

- 160 types
- 10'200 lines of code
- 650 methods

The Java adaptation of the single machine total weighted earliness-tardiness algorithm has the following metrics:

- 40 types
- 3'800 lines of code
- 350 methods

The processing environment is a quad-core, 8-threads Intel Core-I7 machine. The program needs less than 128MB of heap memory to run. The Java virtual machine is the Sun J2SE version 1.6.18. Since each time slice is independent, the computation of the indicators are parallelism at that level. Thus, the indicators of 8 time slices are computed simultaneously and aggregated asynchronously.
9 Simulated case
Since service providers do not provide service evolution probability distributions, we have created a simulated case for which we have invented an organization and all necessary elements for the model to be applied as described in section 8.1. We will then follow the organization's virtual IT manager that will be using the model to assess and reduce the probable costs due to the probable evolution shocks his organization is facing.

Thus, we will first describe the organization and all the elements necessary for the model, the dependencies between the organization's applications and services, and the daily unavailability costs of each application and provided service.

We will then follow the virtual manager as he uses the model iteratively to reduce these probable costs through management decisions. In this case, it will take 3 iterations to eliminate almost all probable future costs induced by evolution shocks.

9.1 Initial situation

9.1.1 Organization setup
We consider an organization with:

- 12 used exogeneous services, from which:
  - 9 services not having received yet a termination warning;
  - 3 services having already received a termination warning.
- 13 internal applications;
- 5 provided services to other organizations.
### 9.1.1.1 Exogeneous services

<table>
<thead>
<tr>
<th>Service id</th>
<th>Termination period</th>
<th>Estimated time to adapt</th>
</tr>
</thead>
<tbody>
<tr>
<td>service_1</td>
<td>40</td>
<td>20</td>
</tr>
<tr>
<td>service_2</td>
<td>40</td>
<td>10</td>
</tr>
<tr>
<td>service_3</td>
<td>30</td>
<td>40</td>
</tr>
<tr>
<td>service_4</td>
<td>60</td>
<td>30</td>
</tr>
<tr>
<td>service_5</td>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>service_6</td>
<td>30</td>
<td>28</td>
</tr>
<tr>
<td>service_7</td>
<td>60</td>
<td>26</td>
</tr>
<tr>
<td>service_8</td>
<td>60</td>
<td>30</td>
</tr>
<tr>
<td>service_9</td>
<td>30</td>
<td>22</td>
</tr>
<tr>
<td>Currently Phased Out - service_10</td>
<td>30</td>
<td>4</td>
</tr>
<tr>
<td>Currently Phased Out - service_11</td>
<td>60</td>
<td>8</td>
</tr>
<tr>
<td>Currently Phased Out - service_12</td>
<td>30</td>
<td>20</td>
</tr>
</tbody>
</table>

### 9.1.1.2 Internal applications

<table>
<thead>
<tr>
<th>Application id</th>
<th>Daily unavailability cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>app_1</td>
<td>10</td>
</tr>
<tr>
<td>app_2</td>
<td>30</td>
</tr>
<tr>
<td>app_3</td>
<td>10</td>
</tr>
<tr>
<td>app_4</td>
<td>20</td>
</tr>
<tr>
<td>app_5</td>
<td>60</td>
</tr>
<tr>
<td>app_6</td>
<td>40</td>
</tr>
<tr>
<td>app_7</td>
<td>10</td>
</tr>
<tr>
<td>app_8</td>
<td>100</td>
</tr>
<tr>
<td>app_9</td>
<td>10</td>
</tr>
<tr>
<td>app_10</td>
<td>10</td>
</tr>
<tr>
<td>app_11</td>
<td>10</td>
</tr>
<tr>
<td>app_12</td>
<td>10</td>
</tr>
<tr>
<td>app_13</td>
<td>10</td>
</tr>
</tbody>
</table>
9.1.1.3 Provided services to other organizations

<table>
<thead>
<tr>
<th>Application id</th>
<th>Daily unavailability cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>offered_1</td>
<td>300</td>
</tr>
<tr>
<td>offered_2</td>
<td>250</td>
</tr>
<tr>
<td>offered_3</td>
<td>100</td>
</tr>
<tr>
<td>offered_4</td>
<td>400</td>
</tr>
<tr>
<td>offered_5</td>
<td>50</td>
</tr>
</tbody>
</table>

9.1.1.4 Organization dependencies

Figure 144. Organization's dependency graph

9.1.1.5 Service evolution probability distributions

All exogeneous services have evolution probability distributions provided by their respective service providers, except for the services service_10, service_11 and service_12 that already have received a termination warning and for which the probability distribution is modeled as having 100% at time = t_{now} and 0 for all other times.
Probability distribution for service_4

Probability distribution for service_5

Probability distribution for service_6
9.2 Assessment
Since the virtual IT manager is in the organization for more than 15 years now, he is one of the few ones who really knows all the dependencies between the organization's applications, the offered services and the used exogeneous services. Therefore, he modeled these dependencies as shown in Figure 144. He also asked his staff to use their task timing system to find out how much time they took to implement these services within the internal applications. This way, he could make an estimation of the time needed to adapt the applications in case of a service evolution event.

The virtual manager then retrieves the service evolution probability distribution from each service provider by using a system described in section 7.5.5. He also asks the sales department to provide him the list of all service contracts that the organization has as a service provider with other organizations. This is to estimate the total daily cost that would be induced by the unavailability of each provided service to other organizations. The manager also asks the heads of his organization's other departments to estimate the cost of unavailability of the internal applications they are using. He then aggregates all that information and defines the elements of the model.

He chooses to use the probable costs indicator because he believes that the most important factor on which his own yearly performance will be judged is the amount of losses due to the unavailability of his organization's information systems. He believes that this is the case whether it concerns only internal applications, or even worse, when it concerns services provided to clients.

He enters all necessary information in the software tool and runs the computations. The system produces a set of graphs that help him understand the probable costs the organization will face because of the unavailability of the internal applications and provided services to other organizations. These graphs are of different types:

- **Type 1:** The probable future costs for each service caused by the non-adaptation of the organization's applications to that service. This graph depicts all the probable costs that will happen in the future but that originate for events at a certain time $t$.
- **Type 2:** The probable costs over time for each service caused by the non-adaptation of the organization's applications to that service. This graph depicts when the probable costs will take place over time.
- **Type 3:** The cumulative costs over time for each service.
- **Type 4:** The total sum of all probable future costs.
- **Type 5:** The total sum of all probable costs over time.
- **Type 6:** The total sum of all cumulative costs over time.
Managing evolution risk of cross-organizational services
Managing evolution risk of cross-organizational services

![Graph](image)
Managing evolution risk of cross-organizational services
Managing evolution risk of cross-organizational services
9.3 Increasing the allocated resources

The virtual manager observes that the organization faces heavy probable losses in the future because the organization will not be able to adapt its applications to the evolving services rapidly enough when they change. The manager also observes that there are few moments in the foreseeable future for which no losses are projected to originate as shown in Figure 145.

![Figure 145. Losses originate almost at any time of the foreseeable future](image)

This means for the manager that he does not have \textit{structurally} enough resources for adapting the applications to the evolving services. There is no single service that is solely responsible for the future costs. This means that the IT manager should ask his top management to provision more resources for the evolutive maintenance of the exogeneous services.

To get an idea of how many more resource would be necessary to reduce the probable loss, the manager will run another simulation with a specified resource level that is twice the actual one. By doing this, the system divides by two the estimated time needed for the adaptation of each service.
Managing evolution risk of cross-organizational services

**Probable future costs at time t for service_7**

**Probable future costs at time t for service_8**

**Probable future costs at time t for service_9**
Managing evolution risk of cross-organizational services

Cumulative probable future costs at time t for service_1

Cumulative probable future costs at time t for service_2

Cumulative probable future costs at time t for service_3
9.4 Renegotiating the contract

The virtual IT manager is pleased to observe that the cumulative probable costs that the organization will face has fallen from 60'000 to 125 when doubling the resources allocated to evolutive maintenance of exogeneous services. The decision to do this will be the trade-off between the cost of the additional resources and the probable costs if not increasing them. This decision will be made by the organization's top management. But the manager has now a way of showing what is at stake because of these services.

The manager also observes that there are only 2 services that are the cause of the remaining costs: service_5 and service_6. He wonders if he could also get rid of these. These services have only probable costs originating at some precise periods in the future as shown in Figure 146.

He would like to know if he could renegotiate the termination period specified in the contracts for these two services with each of their respective providers to keep the probable costs even further down.

He will then change the termination period for both services in the software tool:

- The termination period of service_5 is extended from 15 to 45 days.
- The termination period of service_6 is extended from 30 to 45 days.

He runs again the simulation.
Managing evolution risk of cross-organizational services

Probable costs over time for service_1

Probable costs over time for service_2

Probable costs over time for service_3
Managing evolution risk of cross-organizational services

Cumulative probable future costs at time t for service_7

Cumulative probable future costs at time t for service_8

Cumulative probable future costs at time t for service_9
Managing evolution risk of cross-organizational services
9.5 Refining the contractual adjustment

The IT manager observes that the cumulative costs have sunken from 125 to 17 with no probable costs any more for service_5. The contract renegotiation of the termination period for service_5 from 15 to 45 days is worth 50. The manager also remarks that service_6 still has some little probable costs. He repeats the precedent operation but this time, he changes the termination period of service_6 from 45 to 60 days and runs again the simulation.
This time the probable cumulative costs altogether do not exceed 5. Thus, the value of a contract renegotiation for service_6 from 30 days to 60 days is worth $65 - 5 = 60$. Thus, to eliminate almost completely the organization's total cumulative probable costs induced by the non-adaptation of the applications to the evolution of its used exogeneous services within their contractual time boundaries, the manager can propose the following actions:

- Double the allocated resources dedicated to the evolutive maintenance of exogeneous services.
- Renegotiate the termination period of service_5 from 15 to 45 days for a maximum price of 50.
- Renegotiate the termination period of service_6 from 30 to 60 days for a maximum price of 60.

9.6 Findings

9.6.1 Resource shortage and costs

The proposed model can identify a shortage relative to the resource allocation for the evolutive maintenance of used exogeneous services. The model can also estimate when this shortage will take place and estimate the probable costs that will be caused by the evolution of the used exogeneous services. This permits to price the extension of the termination period specified in a service contract. This price is organization-dependent because the unavailability cost of the internal applications and the provided services to other organizations is specific to each organization. With this model, it is also possible to show that a management mission will not be successful, despite any management proficiency of the individual.
9.6.2 Structural versus contextual shortage
The proposed model permits to identify not only if there is a resource shortage, but moreover if this shortage is structural or contextual. A structural resource shortage is observed when the probable costs are continuously originating at any moment over time as seen in the first iteration of the example. To the contrary, a contextual resource shortage is localized within a limited time period and can be dealt with through renegotiating the termination period of service contracts as done in the second iteration of the example.

9.6.3 Reduction of key-man dependencies
It is important to note that if this model would be used by an organization, it would greatly reduce the dependency the organization has on some of its key people. The IT environment of large organizations has become increasingly so complex, that only a few key individuals know the dependencies between the different applications and the impact of the unavailability of these applications. These key people know by experience what is critical in the information system of their organizations and either they take the decisions on maintenance tasks themselves or they are consulted by the manager in charge for any decision around maintenance. As time passes and the number of applications and dependencies between applications increases, organizations become more and more dependent on these key people. If the proposed model would be applied, these individuals would not be as critical as they are today within their organizations. In a sense, the knowledge that these people have would be transferred to their organization.
10 Limitations and weaknesses

10.1 Service evolution probability distributions
The model uses the hypothesis that the service evolution probability distributions are given by the service providers and that this information is reliable. Thus, the mean of the distribution of the differences between the most probable times of the termination warnings and the realized times of the termination warnings should be close to zero when considering a large number of exogeneous services. Since the service providers do not provide currently this information, this can not be verified.

10.2 Adaptation time
The model needs the estimated time to adapt applications in case of the evolution of a exogeneous service. This adaptation time depends on the nature of the evolution, on the particularities of the applications using the service, on the skills and knowledge of the staff that will perform the adaptations and on the accuracy of the estimation itself. All these factors can lead to underestimate or overestimate the time that is really needed for an adaptation to be performed.

10.3 Dependency identification
It may be difficult to identify all dependencies between the applications within large organizations. The reasons for this may be the lack of documentation, the staff turnover or even the retention of information by the individuals that have it. Hence, many of the dependencies are only discovered or rediscovered when an element in the dependency graph becomes unavailable. Thus, it may be difficult to create the organization's dependency graph.

10.4 Unavailability costs
The daily unavailability costs are difficult to estimate, especially for internal applications. The unavailability costs of the provided services should be more easy to estimate since there are usually established contracts for cross-organizational services.

10.5 Computation time
Since the indicators rely on a cost-based task ordering, we are dependent on the performance of the earliness-tardiness algorithms. The time to compute this order increases exponentially as the number of exogeneous services increases. Thus, the model is functional for the number of exogeneous services on the order of magnitude showed in the example, but may not be useable for a much larger number of services unless we accept to lower the precision of the estimate.
11 Contribution, discussion and conclusion

11.1 Contribution

11.1.1 Overview of dynamic reconfiguration of distributed systems, cross-organizational interactions, services and service contracts
This thesis presented an overview distributed computing in the context of cross-organizational interactions. Distributed systems and the technologies to create them have been presented. The current approaches of dynamic reconfiguration of distributed systems have been reviewed. We have highlighted the usage of services for cross-organizational interactions. The need for services and service contracts have been discussed in detail. Different contract-based approaches have been described.

11.1.2 Presentation of the inadequacy of distributed system technologies applied to cross-organizational interactions
We have presented the reasons why technologies and techniques used in distributed systems and dynamic reconfiguration of distributed systems can't be applied in a cross-organizational context. The first one is the heterogeneity of products and technologies used by the different organizations that interact with each other. The second one is because there is no single organizational authority, responsibility in a cross-organizational interaction. Thus, we state that there will never be automatic upgrades of client applications using evolving cross-organizational services.

11.1.3 Definition of the service evolution challenge
We defined the real service evolution challenge as to determine how much resources an organization needs, and also when these resources will be needed in the future, to adapt its applications that use evolving services of other organizations. We state that these adaptations should be performed in a timely manner such that the applications using these evolving services continue to function without any disruptions. If the disruption-free target is not achievable, the adaptations should be done in such a way that the consequences of these disruptions are minimized.
11.1.4 Proposal of a model

We propose a model to determine how many resources an organization needs to adapt to its evolving world and also when these resources are needed. Using the model, it is possible to:

- Estimate the probability of disruptions induced by the evolution of used exogeneous services.
- Estimate the probable resource shortage to avoid disruptions induced by the evolution of used exogeneous services.
- Identify the amount of resource shortage.
- Estimate the probable cost induced by the resource shortage.
- Estimate if the probable resource shortage is contextual or structural.

Through an example, we show how the model can be used to identify actions that can reduce the probable costs of future evolution shocks faced by an organization. We also show how an organization can price the extension of the termination periods of its used exogeneous services.

11.2 Discussion

The ability to manage IT operations is often associated with the ability to keep their costs as low as possible. The IT managers in charge of maintaining the applications up and running must very often do it on a fixed-budget basis. They experience difficulties to convince their higher management that resources are not sufficient for the expected workload. On the other side, top management has little knowledge of these problems because of the complexity of the information systems and is therefore reluctant to allocate more resources for evolutive maintenance. The proposed model provides a common ground that can act as the basis for constructive discussions on the amount of resources needed to perform evolutive maintenance. This model can help management of different hierarchical levels to communicate more easily with each other. It synthesizes information from a lower operational level into clear indicators for the upper management levels. Hence, it can be viewed as a strategic decision tool that may prevent from potential damages when used wisely.
11.3 Conclusion

In this thesis, we have presented an overview of many technologies and approaches used in distributed computing. We have observed that even if cross-organizational interactions seem to present a lot of similarities with distributed systems, they differ completely in the context of their evolutive maintenance. We have highlighted in section 5.4.3 that, in an cooperative world of sovereign systems, planning and resource allocation are the key drivers behind the continuous and smooth operations of organizations. Technology is secondary. Thus, it is crucial for an organization to know how many resources it will need to follow the pace of evolution of its external world. The level of needed resources for adapting itself to the evolving services it depends on will directly impact the organization's ability to pursue new projects. This is because the resources needed to adapt the applications to its evolving exogeneous services will be taken from the total resource pool of the organization, which includes the resources for new projects. Thus, evolutive maintenance will divert more or less resources from new projects.

These new projects are crucial to maintain the organization's competitiveness relative to the other existing organizations. Hence, if an organization would like to stay competitive in a more and more interconnected world, it will have to minimize the resources needed to maintain its operations that depend on other organizations. The more an organization has available resources for new projects, the more it has chances to create something that may have value in the future.

In this context, our model can help achieve this target. Under the hypothesis that service providers estimate well the timing of the future changes of their services and if this information is easily available, then the model presented in this thesis would permit a better management of the resources needed to adapt an organization to its evolving world. Therefore, the organizations would probably have more available resources to fund new projects. In that sense, the model presented in this thesis is a tool to optimize the resource ratio between the future of an organization and its present.
Appendices

Appendix A : SOAP Example

Since a SOAP message is XML-based, it is easy to provide a readable example. Let's suppose that a stock exchange wants to provide a service that returns the real-time stock quote for a company. As shown in Figure 147, the SOAP message of a request for the IBM stock quote could be similar to the following XML string:

```xml
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <h:from xmlns:h="http://www.stockexchange.com/Header">client@bank.com</h:from>
  </env:Header>
  <env:Body>
    <se:GetStockQuote xmlns:se="http://www.stockexchange.com/feeds/">
      <se:ticker>ibm</se:ticker>
    </se:GetStockQuote>
  </env:Body>
</env:Envelope>
```

Figure 147. SOAP message request example

In this message, the message sender has added a SOAP header that could be used by the receiving endpoint to record the e-mail address of the sender such that the stock exchange can send him through e-mail any information about an IBM future stock event. This message is then received by the stock exchange endpoint which sends back a SOAP message. As shown in Figure 148, the answer of the stock exchange could be in the following form:

```xml
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Body>
    <se:GetStockQuoteResponse xmlns:se="http://www.stockexchange.com/feeds/">
      <se:ticker>ibm</se:ticker>
      <se:quote>94.43</se:quote>
    </se:GetStockQuoteResponse>
  </env:Body>
</env:Envelope>
```

Figure 148. SOAP message response example

There are many other aspects in the SOAP specification. For more information about SOAP please refer to [64].
Appendix B : RM-ODP

The RM-ODP is a joint effort by the International Standards Organization and the International Telecommunications Union (ITU) to provide a coordinating framework for the standardization of Open Distributed Processing (ODP) by creating an architecture which supports distribution, interworking, interoperability and portability. The RM-ODP uses the concept of viewpoints to abstract or describe ODP systems. A viewpoint is a subdivision of the specification of a complete system, established to bring together those particular pieces of information relevant to some particular area of concern during the design of the system. In the RM-ODP ([96], [91]), there are five viewpoints:

- The enterprise viewpoint: A viewpoint on the system and its environment that focuses on the purpose, scope and policies for the system.
- The information viewpoint: A viewpoint on the system and its environment that focuses on the semantic of the information and the information processing performed.
- The computational viewpoint: A viewpoint on the system and its environment that enables distribution through functional decomposition of the system into objects which interact at interfaces.
- The engineering viewpoint: A viewpoint on the system and its environment that focuses on the mechanisms and functions required to support distributed interaction between objects in the system.
- The technology viewpoint: A viewpoint on the system and its environment that focuses on the choice of technology in that system.

For more information about the RM-ODP, please refer to [96] and [91].
Bibliography


[18] Stefanie Rinderle, Manfred Reichert, Peter Dadam, Supporting Workflow Schema Evolution By Efficient Compliance Checks, Ulmer Informatik-Berichte Nr. 2003-02, Computer Science Faculty, University of Ulm, Germany


Managing evolution risk of cross-organizational services


[31] Cory Vondrak, Message-Oriented Middleware, 1997


[34] Walter Hurst, Critical Decisions for Service Oriented Architectures, Web Services Journal, Volume03, Issue 02


[38] Cory Vondrak, Remote Procedure Call, Carnegie Mellon Information technology institute, June 1997


[46] Sun Microsystems, Java Message Service 1.1 Specification, April 2002


Distributed Objects and Applications, LNCS, 2005


[55] Sung-Wook Ryu, B. Clifford Neuman, Garbage Collection for Distributed Persistent Objects, University of Southern California, Workshop on Compositional Software Architectures 1998


[57] Karsten Wolke, Higher Availability of Services in Heterogeneous Distributed Systems, Phd Thesis, Department of Computer Science and Engineering, De Montfort University, United Kingdom, 2007


[59] Randall Perrey, Mark Lycett, Service-Oriented Architecture, Department of Information Systems and Computing, Brunel University

[60] D. Sprott, Service Oriented Process Matters, CBDi Forum Newsletter, 2002

[61] Luciano Baresi, Reiko Heckel, Sebastian Thöne, Dániel Varró, Modeling and Validation of Service Oriented Architectures: Application vs. Style, ESEC/FSE’03, September 1–5, 2003, Helsinki, Finland

[62] Michael P. Papazoglou, Paolo Traverso, Schahram Dustdar, Frank Leymann,
Managing evolution risk of cross-organizational services


[63] Mehran Nikoo, Service-Oriented Architecture. Where Do We Stand?, Dunstan Thomas Consulting


[66] W3C, Web Services Description Language (WSDL) 1.1, March 2001


[69] Mario Jeckle, Barbara Zengler, Active UDDI - an Extension to UDDI for Dynamic and Fault-Tolerant Service Invocation, DaimlerChrysler Research and Technology, 2002


Managing evolution risk of cross-organizational services


[78] Mike Bray, Middleware, Carnegie Mellon Information Technology Institute, January 1997


[82] Christine R. Hofmeister, Dynamic Reconfiguration of Distributed Applications, Doctoral Dissertation, University of Maryland, USA, 1993


[87] Christophe Bidan, Valérie Issarny, Titos Saridakis, Apostolos Zarras, A Dynamic Reconfiguration Service for CORBA, Proceedings of the International Conference on Configurable Distributed Systems, March 04 - 06, 1998, Annapolis, Maryland, USA


[93] Lauren A. Tewksbury, Louise E. Moser, Peter M. Melliar-Smith, Live Upgrades of CORBA Applications Using Object Replication, Proceedings of the IEEE International Conference on Software Maintenance (ICSM'01)


[95] Sameer Ajmani, Barbara Liskov, Scheduling and Simulation: How to Upgrade Distributed Systems, Massachusetts Institute of Technology, Laboratory for Computer Science, 2003


Managing evolution risk of cross-organizational services

[98] BusinessDictionary.com, Organization, Online definition


Managing evolution risk of cross-organizational services


2000


[123] Lea Kutvonen, Automated Management of Inter-Organisational Applications, Proceedings of the Sixth International International Enterprise Distributed Object Computing Conference (EDOC’02)


[125] Louise E. Moser, Peter M. Melliar-Smith, Lauren A. Tewksbury, Online Upgrades Become Standard, Proceedings of the 26 th Annual International Computer Software and Applications Conference (COMPSAC’02)


[133] Thomas Reicher, Asa MacWilliams, Bernd Bruegge, Dependency Graph Pattern, Ninth European Conference on Pattern Languages of Programs 07th - 11th July 2004, Irsee, Germany


Erfurt, Germany


