NORTH ATLANTIC TREATY
ORGANISATION

RESEARCH AND TECHNOLOGY
ORGANISATION

AC/323(IST-064)TP/256

www.rto.nato.int

RTO MEETING PROCEEDINGS **MP-IST-064**

Building Robust Systems with Fallible Construction

(Bâtir des systèmes sûrs à partir de constructions faillibles)

This Report documents the material presented at the IST-064/RWS-011 Workshop held in Prague, Czech Republic, 9-10 November 2006.

Published May 2009

Distribution and Availability on Back Cover

NORTH ATLANTIC TREATY
ORGANISATION

RESEARCH AND TECHNOLOGY
ORGANISATION

AC/323(IST-064)TP/256

www.rto.nato.int

RTO MEETING PROCEEDINGS **MP-IST-064**

Building Robust Systems with Fallible Construction

(Bâtir des systèmes sûrs à partir
de constructions faillibles)

This Report documents the material presented at the IST-064/RWS-011
Workshop held in Prague, Czech Republic, 9-10 November 2006.

The Research and Technology Organisation (RTO) of NATO

RTO is the single focus in NATO for Defence Research and Technology activities. Its mission is to conduct and promote co-operative research and information exchange. The objective is to support the development and effective use of national defence research and technology and to meet the military needs of the Alliance, to maintain a technological lead, and to provide advice to NATO and national decision makers. The RTO performs its mission with the support of an extensive network of national experts. It also ensures effective co-ordination with other NATO bodies involved in R&T activities.

RTO reports both to the Military Committee of NATO and to the Conference of National Armament Directors. It comprises a Research and Technology Board (RTB) as the highest level of national representation and the Research and Technology Agency (RTA), a dedicated staff with its headquarters in Neuilly, near Paris, France. In order to facilitate contacts with the military users and other NATO activities, a small part of the RTA staff is located in NATO Headquarters in Brussels. The Brussels staff also co-ordinates RTO's co-operation with nations in Middle and Eastern Europe, to which RTO attaches particular importance especially as working together in the field of research is one of the more promising areas of co-operation.

The total spectrum of R&T activities is covered by the following 7 bodies:

• AVT	Applied Vehicle Technology Panel
• HFM	Human Factors and Medicine Panel
• IST	Information Systems Technology Panel
• NMSG	NATO Modelling and Simulation Group
• SAS	System Analysis and Studies Panel
• SCI	Systems Concepts and Integration Panel
• SET	Sensors and Electronics Technology Panel

These bodies are made up of national representatives as well as generally recognised 'world class' scientists. They also provide a communication link to military users and other NATO bodies. RTO's scientific and technological work is carried out by Technical Teams, created for specific activities and with a specific duration. Such Technical Teams can organise workshops, symposia, field trials, lecture series and training courses. An important function of these Technical Teams is to ensure the continuity of the expert networks.

RTO builds upon earlier co-operation in defence research and technology as set-up under the Advisory Group for Aerospace Research and Development (AGARD) and the Defence Research Group (DRG). AGARD and the DRG share common roots in that they were both established at the initiative of Dr Theodore von Kármán, a leading aerospace scientist, who early on recognised the importance of scientific support for the Allied Armed Forces. RTO is capitalising on these common roots in order to provide the Alliance and the NATO nations with a strong scientific and technological basis that will guarantee a solid base for the future.

The content of this publication has been reproduced
directly from material supplied by RTO or the authors.

Published May 2009

ISBN 978-92-837-0081-4

Table of Contents

Chapter 6 – Unresolved Challenges

List of Participants

CANADA

W. Morven GENTLEMAN
Faculty of Computer Science
Dalhousie University
Halifax, Nova Scotia B3H 4R2
Email: Morven.Gentleman@dal.ca
Professor – Task Team Chair

Frédéric PAINCHAUD
Knowledge & Information Management
Defence Research and Development Canada
2459 boul. Pie-XI Nord
Val-Bélair, Québec G3J 1X5
Email: Frederic.Painchaud@drdc-rddc.gc.ca
Defence Scientist – Expertise: Defence Systems

CZECH REPUBLIC

Tomas FEGLAR
Vondrousova 1199
163 00 Prague 6
Email: feglar@centrum.cz
Computer Science Consultant – Expertise: Process Integration and Systems Engineering

Milan SNAJDER
Military Technology Institute of Air Force
VTULaPVO
Mladoboleslavska 944
197 21 Prague 97
Email: milan.snajder@vtui.cz
Professor – Task Team Member

FRANCE

Christophe DONY
Université de Montpellier
LIRMM
161 rue Ada
34392 Montpellier Cedex 5
Email: dony@lirmm.fr
Researcher – Expertise: Exception Handling

NETHERLANDS

Maarten BOASSON
Faculty of Science
University of Amsterdam
Kruislaan 404
1098 SM Amsterdam
Email: boasson@science.uva.nl
Consultant – Expertise: Software Architecture for Distributed Applications

NETHERLANDS (cont'd)

Yves VAN DE VIJVER
National Aerospace Laboratory (NLR)
Anthony Fokkerweg 2
PO Box 90502
1006 BM Amsterdam
Email: vyver@nlr.nl
Engineer – Task Team Member

UNITED KINGDOM

Alexander ROMANOVSKY
School of Computer Science
The University of Newcastle-upon-Tyne
Newcastle-upon-Tyne, NE1 7RU
Email: alexander.romanovsky@ncl.ac.uk
Professor – Expertise: Software Fault Tolerance

Joe SVENTEK
Department of Computing Science
University of Glasgow
17 Lilybank Gardens
Glasgow, Scotland G12 8RZ
Email: joe@dcs.gla.ac.uk
Professor of Communications Systems – Expertise: Self Managed Systems and Networks

UNITED STATES

Mary SHAW
Institute for Software Research International
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213-3891
Email: mary.shaw@cmu.edu
A.J. Perlis Professor – Expertise: Software Architecture

Building Robust Systems with Fallible Construction

(RTO-MP-IST-064)

Executive Summary

Today's NATO military commanders depend on large, complex software systems that must be more predictable and trustworthy than traditional development methods can deliver for the available time and cost investments. This requirement is not quite compatible with the traditional software development that is prevalent in today's military acquisition methods. Today's systems are typically integrated from components that may themselves contain flaws, originating in specification, design or implementation errors, or in miscommunication between different teams involved in the development. "System of Systems", where components are systems in and of themselves, are a significant factor. More seriously, the integration process itself may be flawed. This situation can arise in the NATO context, for instance, when coalitions are formed quickly, and complex systems must be integrated from subsystems supplied by different nations.

The workshop was organized to review past and present understanding of the challenge, as well as examining relevant approaches to address them. Rather than an exchange of pre-prepared material, the workshop was intended as a working meeting with a goal of producing a deliverable that is a summary of the state of the art.

The workshop topic is related to Software Fault Tolerance, a topic that has been studied at least since 1970. Worldwide much has been learned about how to address those problems, as they were understood at the time. However changes in perspective as to what constitute the challenges, and changes in available and commonplace technology, have led to a need to go beyond conclusions reached in the past.

The proceedings include position statements from the participants, slides from the presentations made by the participants, and the one complete paper that was submitted. Minutes of the discussions provide insight into how the deliverable, the final report of task group IST-047/RTG-019, was shaped.

Bâtir des systèmes sûrs à partir de constructions faillibles

(RTO-MP-IST-064)

Synthèse

Aujourd'hui, les commandants militaires de l'OTAN sont tributaires de systèmes logiciels importants et complexes qui doivent être plus prévisibles et dignes de confiance que ce que peuvent produire les méthodes de développement traditionnelles compte tenu du temps disponible et des coûts d'investissement. Ces impératifs ne sont pas vraiment compatibles avec le développement traditionnel des logiciels qui prévaut actuellement dans les méthodes militaires d'acquisition. Actuellement, les systèmes sont en général intégrés à partir de composants qui peuvent contenir des imperfections provenant des spécifications, d'erreurs de fabrication ou de mise en œuvre, ou d'une mauvaise communication entre les différentes équipes impliquées dans le développement. Les « systèmes de systèmes », dont les composants sont eux-mêmes des systèmes dans le système en sont un élément significatif. Plus sérieusement, le processus d'intégration peut être lui-même défectueux. Cette situation peut advenir dans un cadre OTAN, par exemple quand les coalitions sont formées rapidement et quand des systèmes complexes doivent être intégrés à partir de sous-systèmes fournis par différentes nations.

L'atelier a été organisé pour passer en revue les façons passées et présentes d'appréhender les enjeux mais aussi pour examiner les approches pertinentes pour les aborder. Plutôt que d'échanger des éléments préparés à l'avance, l'atelier a été voulu comme une réunion de travail avec pour but de produire un résumé de ce qu'est l'état de l'art dans le domaine.

Le thème de l'atelier concernait la limite de tolérance acceptable aux défauts des logiciels, sujet étudié depuis au moins 1970. Dans le monde entier, on a beaucoup appris sur la façon de traiter ces problèmes, tels qu'on les appréhendait à l'époque. Cependant, les changements de perspective sur ce qui constitue les enjeux et les évolutions de la technologie disponible courante, ont rendu nécessaire d'aller au delà des conclusions retenues par le passé.

Les rapports contiennent l'exposé des positions des participants, les diapositives des présentations faites par ces participants et le document complet unique qui a été proposé. Les minutes des débats donnent une vision interne de la façon dont le produit délivré, compte-rendu final du groupe opérationnel IST-047/RTG-019 a été élaboré.

Chapter 1 – Introduction and Motivation

W. Morven Gentleman
Professor
Faculty of Computer Science
Dalhousie University
Halifax, Nova Scotia
CANADA

Morven.Gentleman@dal.ca

Today's NATO military systems depend on large, complex software with the need to be built and deployed more rapidly and cheaper than traditional development methods can deliver. Moreover, because military commanders depend on these systems, they must be more predictable and trustworthy than traditional development methods can deliver for the available time and cost investments. However this requirement is not quite compatible with the traditional project-oriented view of software development, which is prevalent in today's military acquisition methods.

Today's systems are typically integrated from components. These components may themselves contain flaws, originating in specification, design or implementation errors, or in miscommunication between different teams involved in the development. More seriously, the integration process itself may be flawed, as when pre-existing components are used for purposes that their developers had not envisioned, and the integrators misunderstand the detailed behaviour of the components. This situation can arise in the NATO context, for instance, when coalitions are formed quickly, and complex systems must be integrated from subsystems supplied by different nations: the components might be flawed, they might be misused, and the integration might be inappropriately performed.

A different perspective comes from the recognition that components sometimes are systems in and of themselves, and that these systems may not lose their identity when integrated into a "System of Systems" but as well as being expected to fulfill their role in the System of Systems, may continue to retain their original purpose, with their independent management, independent operational needs, and independent evolution [Meier 1999][1]. Frictions, even conflicts, can obviously arise in such mixed circumstances. Interoperability failures between different national systems often are of this form.

Experience with interoperating commercial products, especially in the context of the Internet, indicates that robustness to fallible components and fallible integration can be achieved without centralized predictive coordination. Appropriate software architecture, redundancy in functional components, and enforcement of critical interface standards appear to be key elements of success. Improved registry and plug-and-play concepts can help automate integration and reduce configuration problems.

If we are to try to build infallible systems with fallible construction methods, there is a need to review the advances in software development in the commercial market. There is also a need to evaluate the requirements of military software development, bring forth lessons to be learned and to identify areas of research and draw projections especially for the procurement community.

RTO IST-064/RWS-11 was a cooperative international workshop specifically aimed at investigating how to usefully work with software that is believed to be faulty. The workshop was intended to be a truly interactive workshop rather than only a mini-conference of presented papers. That is, it was to be a working meeting to produce a specific deliverable: a report summarizing the state of the art. Attendees

[1] [Meier 1999] M. W. Meier, "Architecting Principles for Systems-of-Systems", Systems Engineering, 2:1, 1999.

were expected to participate fully in identifying the critical issues for successfully working with software believed to be faulty, determining community challenge problems, applications and case studies, and helping to set a research agenda for the area. The workshop was structured around a set of topics and issues identified by the participants. Prior to the workshop, all invited participants were asked to submit a position paper outlining their perspective, and to categorize their own position paper and their related work. The categorizations chosen helped to suggest session themes. The workshop consisted of several focused sessions, which set the stage for scheduled and impromptu presentations by participants and follow-up discussion.

The Technical Activity Proposal authorizing this workshop had defined possible topics to be covered:

1) Choices of software architecture for robustness.
2) Integration process and tools.
3) Critical interface standards.
4) Interoperability with complementary or related products.
5) Empirical behaviour investigation through testing.
6) Oracles to ascertain plausibility of results.
7) Scalability of fault recovery.
8) Autonomic systems, self-healing, dynamic re-configuration.
9) Coping with component evolution.
10) Aids to retraining users.
11) Scaffolding reuse.
12) Regression tests, integration tests, integrity testing, consistency testing.
13) Project metric tracking.
14) Implications for cultural change.
15) Architectures for defect tolerance.
16) CORBA reliability.

In planning the workshop, the overlap with software fault tolerance was noted, so topics for discussion were suggested where there was a difference today from situations that had been considered in the past with respect to fault tolerance:

1) Use of pre-existing components.
2) Excess computing capacity that can be used:
   a) For training and simulation to ensure proper functioning when needed;
   b) For audit routines to monitor integrity of run-time and persistent data structures; and
   c) For self-management (AI techniques).
3) Decentralized operation and control.
4) Conflicting, unknowable, diverse requirements.
5) Continuous evolution and deployment.
6) Heterogeneous, inconsistent, changing elements.
7) Indistinct people/system boundary.

8) Failures considered normal.
9) New forms of acquisition and policy.
10) Use of signatures to identify potential upcoming problems:
    a) Correlation only yet causality may be needed to be able to take corrective actions.
11) Depth of exposure of commanders to computers.
12) Display of uncertainty.
13) Size and level of expertise of development teams.
14) Increasing use of concurrency (and lack of exception handling mechanisms for concurrency).
15) Voting mechanisms may not work (properly) because:
    a) Answers not always being unique or uniquely represented;
    b) Answers that cannot be compared automatically; or
    c) Answers that cannot guarantee to be compared within certain time limits.
16) Classical engineering use linear models:
    a) What does linearity and continuity mean for software systems?
    b) Can we build linear models for software?
    c) If not, can we handle non-linear models?
17) Agent-based architectures and the publish/subscribe mechanism decouple components. Does this make the understanding of the system as a whole not more difficult?

Some new opportunities for research were also suggested:

1) Revise/reopen previous research (e.g. strongly guarded data structures).
2) New ways of doing distributed computing.
3) Information synthesis/fusion (area also covered in grid computing field).
4) Taking sociological issues into account.

Unfortunately, it proved impossible to arrange participation in the workshop by any representative of one obvious research group relevant to the topic, the Stanford-UC Berkeley initiative on Recovery Oriented Computing. This group has explicitly taken the position that failures are inevitable, if only because of inappropriate input by humans such as system operators as well as end-users. Consequently, their approach has been that rather than concentrating on avoiding failure, attention should be focused on making recovery from failure faster and more reliable. Perhaps surprisingly, there is as yet very little other work from this perspective. In the absence of a representative of this group, the workshop discussions deliberately had to attempt to anticipate the reactions this group would have.

The original intent was that the report would include:

1) Summary of workshop results;
2) Position papers;
3) An identified research agenda;
4) Community challenge problems; and
5) Plans for future cooperative activities.

2.1 – Architectural Support for Integration in Distributed Reactive Systems

Maarten Boasson
Quaerendo Invenietis bv
Universiteit van Amsterdam
The Netherlands
maarten@quaerendo.com

Abstract

Due to the many possible interactions with an ever changing environment, combined with stringent requirements regarding temporal behaviour, robustness, availability, and maintainability, large-scale embedded systems are very complex in their design. Coordination models offer the potential of separating functional requirements from other aspects of system design. In this paper we present a software architecture for large-scale embedded systems that incorporates an explicit coordination model. Conceptually the coordination model consists of application processes that interact through a shared data space - no direct interaction between processes is possible. Starting from this relatively simple model we derive successive refinements of the model to meet the requirements that are typical for large-scale embedded systems.

The software architecture has been applied in the development of commercially available command-and-control and traffic management systems. Experience shows that due to the very high degree of modularity and the maximal independence between modules, these systems are relatively easy to develop and integrate in an incremental way. Moreover, distribution of processes and data, fault-tolerant behaviour, graceful degradation, and dynamic reconfiguration are directly supported by the architecture.

1. Introduction

Due to the many possible interactions with an ever changing environment, combined with stringent requirements regarding temporal behaviour, robustness, availability, and maintainability, large-scale embedded systems, like traffic management, process control, and command-and-control systems, are very complex in their design. The tasks performed by these systems typically include: (1) processing of measurements obtained from the environment through sensing devices, (2) determination of model parameters describing the environment, (3) tracking discrepancies between desired state and perceived state, (4) taking corrective action, and (5) informing the operator or team of operators about the current and predicted state of affairs. All tasks are very closely related and intertwined, and particularly in large-scale systems, there is a huge number of model parameters, which are often intricately linked through numerous dependencies. It is therefore a very natural approach to design the software for such systems as a monolithic entity, in which all relevant information (deductive knowledge and actual data) is readily accessible for all the above mentioned parts.

There is, however, a strong and well-known reason to proceed differently: a software system thus conceived is very difficult to implement, and even more difficult to modify should the purpose of the system be changed, or the description of the environment be refined. Adopting a *modular* approach to design, the various functions implemented in software are separated into different modules that have some independence from each other. Such an approach - well established today as standard software engineering practice - leads to better designs, and reduces development time and the likelihood of errors.

Unfortunately, with today’s highly sophisticated systems, this is still not good enough. In addition to the functional requirements of these systems, many non-functional requirements, such as a high degree of availability and robustness, distribution of the processing over a possibly wide variety of different host processors, and (on-line) adaptability and extendibility, place constraints on the design freedom that can hardly be met with current design approaches. A methodology for the design of large-scale distributed embedded systems should provide (a basis for) an integral solution for the various types of requirements. Traditional design methods based on functional decomposition are not adequate. The sound principle of modularity needs therefore to be further exploited to cover non-functional requirements as well.

Recently coordination models and languages have become an active area of research [6]. In [7] it was argued that a complete programming model consists of two separate components: the computation model and the coordination model. The computation model is used to express the basic tasks to be performed by a system, i.e. the system's functionality. The coordination model is applied to organize the functions into a coherent ensemble; it provides the means to create processes and facilitates communication. One of the greater merits of separating computation from coordination is the considerably improved modularity of a system. The computation model facilitates a traditional functional decomposition of the system, while the coordination model accomplishes a further decoupling between the functional modules in both space and time. This is exemplified by the relative success of coordination languages in the field of distributed and parallel systems.

Since the early 80's we have developed and refined a software architecture for large-scale distributed embedded systems [2], that is based on a separation between computation and coordination. Below, we first present the basic software architecture, after which we shall focus on the underlying coordination model. We demonstrate how the basic coordination model can be gradually refined to include non-functional aspects, such as distributed processing and fault-tolerance, in a modular fashion. The software architecture has been applied in the development of commercially available command-and-control, and traffic management systems. We conclude with a discussion of our experiences in the design of these systems.

2. Software architecture

A software architecture defines the organisational principle of a system in terms of types of components and possible interconnections between these components. In addition, an architecture prescribes a set of design rules and constraints governing the behaviour of components and their interaction [4]. Traditionally, software architectures have been primarily concerned with structural organisation and static interfaces. With the growing interest in coordination models, however, more emphasis is placed on the organizational aspects of behaviour and interaction.

In practice, many different software architectures are in use. Some well-known examples are the Client/Server and Blackboard architectures. Clearly, these architectures are based on different types of components - clients and servers versus knowledge sources and blackboards - and use different styles of interaction - requests from clients to servers versus writing and reading on a common blackboard.

The software architecture, named SPLICE, that we developed for distributed embedded systems basically consists of two types of components: *applications* and a *shared data space*. Applications are active, concurrently executing processes that each implement part of the system's overall functionality. Besides process creation, there is no direct interaction between applications; all communication takes place through a logically shared data space simply by reading and writing data elements. In this sense SPLICE bears strong resemblance to coordination languages and models like Linda [5], Gamma [1], and Swarm [9], where active entities are coordinated by means of a shared data space.

2.1. The shared data space

The shared data space in SPLICE is organized after the well-known relational data model. Each data element in the shared data space is associated with a unique *sort*, that defines its structure. A sort definition declares the *name* of the sort and the *record fields* the sort consists of. Each record field has a type, such as integer, real, or string; various type constructors, such as enumerated types, arrays, and nested records, are provided to build more complex types.

Sorts enable applications to distinguish between different kinds of information. A further differentiation between data elements of the same sort is made by introducing identities. As is standard in the relational data model, one or more record fields can be declared as *key* fields. Each data element in the shared data space is uniquely determined by its sort and the value of its key fields. In this way applications can unambiguously refer to specific data elements, and relationships between data elements can be explicitly represented by referring from one data element to the key fields of another.

To illustrate, we consider a simplified example taken from the domain of air traffic control. Typically a system in this domain would be concerned with various aspects about flights, such as flight plans and the progress of flights as tracked from the reports that are received from the system's surveillance radar. Hence, we define sorts *flightplan*, *report*, and *track* as indicated in Figure 1.

Sort *flightplan* declares four fields: a flight number, e.g. KL332 or AF1257, the scheduled time for departure and arrival, and the type of aircraft that carries out the flight, e.g. a Boeing 737 or an Airbus A320. By declaring the flight number as a key field, it is assumed that each flight plan is uniquely determined by its flight number.

Sort *report* contains the measurement vector of an object as returned at a specific time by the system's surveillance radar. The measurement vector typically con-

sort *flightplan*
  key *flightnumber* : string
  departure : time
  arrival : time
  aircraft : string

sort *report*
  key *index* : integer
  measurement : vector
  timestamp : time

sort *track*
  key *flightnumber* : string
  timestamp : time
  state : vector

Figure 1. Sort definitions: an example.

tains position information. A unique index is attached to be able to distinguish between different reports.

Through a correlation and identification process, the progress of individual flights is recorded in sort *track*. The state vector typically contains position and velocity information on the associated flight number, that is computed from consecutive measurements. The timestamp identifies the time at which the state vector has been last updated.

2.2. Applications

Basically, applications interact with the shared data space by writing and reading data elements. SPLICE does not provide an operation for globally deleting elements from the shared data space. Instead, data can be removed implicitly using an overwriting mechanism. This mechanism is typically used to update old data with more recent values as the system's environment evolves over time. Additionally, applications can hide data, once read, from their view. This operation enables applications to progressively traverse the shared dataspace by successive read operations. By the absence of a global delete operation, the shared dataspace in SPLICE models a dynamically changing information store, where data can only be read or written. This contrasts the view where data elements represent shared resources, that can be physically consumed by applications.

SPLICE extends an existing (sequential) programming language with coordination primitives for creating processes and for interacting with the shared dataspace. More formally, the primitives are defined as follows.

- **create**(f): creates a new application process from the executable file named f, and run it in parallel to the existing applications.

- **write**(α, x): inserts an element x of sort α into the shared data space. If an element of sort α with the same key value as x already exists in the shared dataspace, then the existing element is replaced by x.

- **read**(α, q, t): reads an element of sort α from the shared dataspace, satisfying query q. The query is formulated as a predicate over the record fields of sort α. In case a matching element does not exist, the operation blocks until either one becomes available or until the timeout t has expired. If the latter occurs, a timeout error is returned by the operation. The timeout is an optional argument: if absent the read operation simply blocks until a matching element becomes available. In case more than one matching element can be found, one is selected non-determinstically.

- **get**(α, q, t): operates identically to the read operation, except that the element returned from the shared dataspace becomes hidden from the application's view, that is, the same element cannot be read a second time by the application.

The overwriting mechanism that is used when inserting data elements into the shared dataspace potentially gives rise to conflicts. If at the same time two different applications each write a data element of the same sort and with the same key value, one element will overwrite the other in a nondeterministic order. Consequently one of the two updates will be lost. In SPLICE this type of nondeterministic behaviour is considered undesirable. The architecture therefore imposes the design constraint that for each sort at most one application shall write data elements with the same key value.

As an illustration we return to the air traffic control example from the previous section. Consider an application process that tracks the progress of flight number n. This application continuously reads new reports from the surveillance radar and updates the track data of flight number n accordingly. The application process can be defined as indicated by the code fragment in Figure 2.

The application first reads the initial track data for flight number n from the shared dataspace. The initial data is produced by a separate application that is responsible for track initiation. The application then enters a loop where it first reads a new report r from the shared dataspace. If the report correlates with the current track t,

```
t := get(track, flightnumber = n);
repeat
  r := get(report, true);
  if correlates(r, t) then
    update(t, r);
    write(track, t);
  end if
until terminated(t);
```

Figure 2. Coordination primitives: an example.

as expressed by the condition *correlates*(*r*, *t*), then track *t* is updated by the newly received report, using the procedure *update*(*t*, *r*). The updated track is inserted into the shared dataspace, replacing the previous track data of flight number *n*. This process is repeated until track *t* is terminated. Termination can be decided, for instance, if a track did not receive an update over a certain period of time.

3. Refinements of the architecture

The shared dataspace architecture is based on an ideal situation where many non-functional requirements, such as distribution of data and processing across a computer network, fault-tolerance, and system response times, need not be taken into account. We next discuss how, through a successive series of modular refinements, a software architecture can be derived that fully supports the development of large-scale, distributed embedded systems.

3.1. A distributed software architecture

The first aspect that we consider here is distribution of the shared data space over a network of computer systems. The basic architecture is refined by introducing two additional components. As illustrated in Figure 3, the additional components consist of *heralds* and a *communication network*.

Each application process interacts with exactly one herald. A herald embodies a local database for storing data elements, and processing facilities for handling all communication needs of the application processes. All heralds are identical and need no prior information about either the application processes or their communication requirements. Communication between heralds is established by a message passing mechanism. Messages between heralds are handled by the communication network that interconnects them. The network must support broadcasting, but should preferably also support direct addressing of heralds, and multicasting. An application process interacts with its assigned herald by means of the interaction primitives from section 2.2. The interaction with heralds is transparent with respect to the shared dataspace model: application processes continue to operate on a logically shared dataspace.

The heralds are passive servers of the application processes, but are actively involved in establishing and maintaining the required inter-herald communication. The communication needs are derived dynamically by the collection of heralds from the read and write operations that are issued by the application processes. The protocol that is used by the heralds to manage communication is based on a *subscription* paradigm that can be briefly outlined as follows.

First consider an application that performs a write operation. The data element is transferred to the application's herald, which initially stores the element into its local database, overwriting any existing element of the same sort and with the same key value.

Next consider an application that issues a read request for a given sort. Upon receipt of this request, the application's herald first checks whether this is the first request for that particular sort. If it is, the herald broadcasts the name of the sort on the network.

All other heralds, after receiving this message, register the herald that performed the broadcast as a *subscriber* to the sort carried by the message. Next each herald verifies if its local database contains any data elements of the requested sort, previously written by its application process, in which case copies of these elements are transferred to the newly subscribed herald. After this initial transfer, any subsequently written data of the requested sort will be immediately forwarded to all subscribed heralds.

Each subscribed herald stores both the initially and all subsequently transferred copies into its local database,

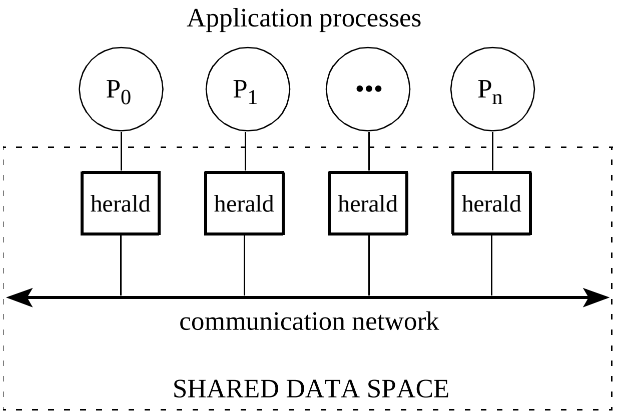


Figure 3. A distributed software architecture.

overwriting any existing data of the same sort and with the same key value. During all transfers a protocol is used that preserves the order in which data elements of the same sort have been written by an application. This mechanism in combination with the architecture's design constraint that for each sort at most one application writes data elements with the same key value, guarantees that overwrites occur in the same order with all heralds. Otherwise, communication by the heralds is performed asynchronously.

The search for data elements matching the query of a read request is performed locally by each herald. If no matching element can be found, the operation is suspended either until new data of the requested sort arrives or until the specified timeout has expired.

Execution of a get operation is handled by the heralds similarly to the read operation, except that the returned data element is removed from the herald's local database.

As a result of this protocol, the shared dataspace is selectively replicated across the heralds in the network. The local database of each herald contains data of only those sorts that are actually read or written by the application it serves. In practice the approach is viable, particularly for large-scale distributed systems, since the applications are generally interested in only a fraction of all sorts. Moreover, the communication pattern in which heralds exchange data is relatively static: it may change when the operational mode of a system changes, or in a number of circumstances in which the configuration of the system changes (such as extensions or failure recovery). Such changes to the pattern are very rare with respect to the number of actual communications using an established pattern. It is therefore beneficial from a performance point of view to maintain a subscription registration. After an initial short phase each time a new sort has been introduced, the heralds will have adapted to the new communication requirement. This knowledge is subsequently used by the heralds to distribute newly produced data to all the heralds that hold a subscription. Since subscription registration is maintained dynamically by the heralds, all changes to the system configuration will automatically lead to adaptation of the communication patterns.

Note that there is no need to group the distribution of a data element to the collection of subscribed heralds into an atomic transaction. This enables a very efficient implementation in which the produced data is distributed asynchronously and the latency between actual production and use of the data depends largely on the consuming application processes. This results in upper bounds that are acceptable for distributed embedded systems where timing requirements are of the order of milliseconds.

3.2. Temporal aspects

The shared dataspace as introduced in section 2, models a persistent store: data once written remains available to all applications until it is either overwritten by a new instance or hidden from an application's view by execution of a get operation. The persistence of data decouples applications in time. Data can be read, for instance, by an application that did not exist the moment the data was written, and conversely, the application that originally wrote the data might no longer be present when the data is actually read.

Applications in the embedded systems domain deal mostly with data instances that represent continuous quantities: data is either an observation sampled from the system's environment, or derived from such samples through a process of data association and correlation. The data itself is relatively simple in structure; there are only a few data types, and given the volatile nature of the samples, only recent values are of interest. However, samples may enter the system at very short intervals, so sufficient throughput and low latency are crucial properties. In addition, but to a lesser extent, embedded systems maintain discrete information, which is either directly related to external events or derived through qualitative reasoning from the sampled input.

This observation leads us to refine the shared dataspace to support volatile as well as persistent data. The sort definition, which basic format was introduced in section 2.1, is extended with an additional attribute that indicates whether the instances of a sort are volatile or persistent. For persistent data the semantics of the read and write operations remain unchanged. Volatile data, on the other hand, will only be visible to the collection of applications that is present at the moment the data is written. Any application that is created afterwards, will not be able to read this data.

Returning to the air traffic control example from Figure 1, the sort *report* can be classified as volatile, whereas the sorts *track* and *flightplan* are persistent. Consequently, the tracking process, as specified in Figure 2, does not receive any reports from the surveillance radar that were generated prior to its creation. After the tracking process has been created, it first gets the initial track data and then waits until the next report becomes available.

Since the initial track data is produced exactly once, the tracking process must be guaranteed to have access to it, otherwise the process might block indefinitely. This implies that the sort *track* must be persistent.

The subscription-based protocol, that manages the distribution of data in a network of computer systems, can be refined to exploit the distinction between volatile and persistent data. Since volatile data is only available to the applications that are present at the moment the data is written, no history needs to be kept. Consequently, if an application writes a data element, it is immediately forwarded to the subscribed heralds, without storing a copy in the application's local database. This optimization reduces the amount of storage that is required. Moreover, it eliminates the initial transfer of any previously written data elements, when an application performs the first read operation on a sort. This enables a newly created application to integrate into the communication pattern without initial delay, which better suits the timing characteristics that are typically associated with the processing of volatile data.

3.3. Fault-tolerance

Due to the stringent requirements on availability and safety that are typical of large-scale embedded systems, there is the need for redundancy in order to mask hardware failures during operation. Fault-tolerance in general is a very complex requirement to meet and can, of course, only be partially solved in software. In SPLICE, the heralds can be refined to provide a mechanism for fault-tolerant behaviour. The mechanism is based on both data and process replication. By making fault-tolerance a property of the software architecture, the design complexity of applications can be significantly reduced.

In this paper we only consider failing processing units, and we assume that if a processor fails, it stops executing. In particular we assume here that communication never fails indefinitely and that data does not get corrupted.

If a processing unit in the network fails, the data that is stored in this unit, will be permanently lost. The solution is to store copies of each data element across different units of failure. The subscription-based protocol described in section 3.1 already implements a replicated storage scheme, where copies of each data element are stored with the producer and each of the consumers. The basic protocol, however, is not sufficient to implement fault-tolerant data storage in general. For instance, if data elements of a specific sort have been written but not (yet) read, the elements are stored with the producer only. A similar problem occurs if the producers and consumers of a sort happen to be located on the same processing unit.

The solution is to store a copy of each data element in at least one other unit of failure. The architecture as depicted in Figure 3 is extended with an additional type of component: a persistent database. This component executes a specialized version of the subscription protocol. On start-up a persistent database broadcasts the name of each persistent sort on the network. As a result of the subscription protocol that is executed by the collection of heralds, any data element of a persistent sort that is written by an application, will be automatically forwarded to the persistent database. There can be one or more instances of the persistent database executing on different processing units, dependent on the required level of system availability. Moreover, it is possible to load two or more persistent databases with disjoint sets of sort names, leading to a distributed storage of persistent data.

When a processing unit fails, also the applications that are executed by this unit will be lost. The architecture can be refined to support both passive and active replication of applications across different processing units in the network.

Using passive replication, only one process is actually executing, while one or more back-ups are kept off-line, either in main memory or on secondary storage. When the processing unit executing the active process fails, one of the back-ups is activated. In order to be able to restore the internal state of the failed process, it is required that each passively replicated application writes a copy of its state to the shared dataspace each time the state is updated. The internal state can be represented by one or more persistent sorts. When a back-up is activated, it will first restore the current state from the shared dataspace and then continue execution.

When timing is critical, active replication of processes is often a more viable solution. In that case, multiple instances of the same application are executing in parallel, hosted by different processing units; all instances read and write data. Typically active replication is used when timing is critical and the failing component must be replaced instantaneously.

The subscription-based protocol can be refined to support active replication transparently. If a particular instance of a replicated application performs a write operation, its herald attaches a unique replication index as a

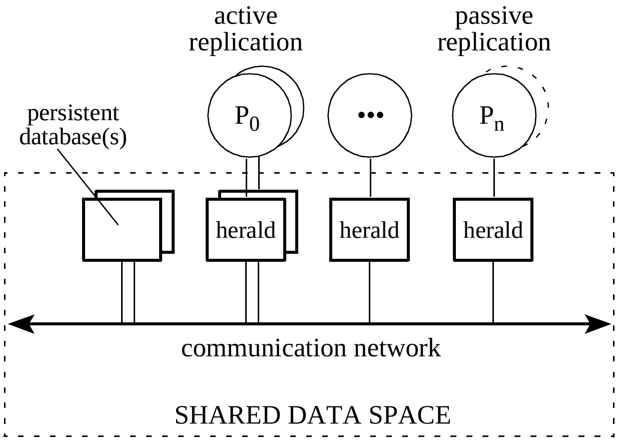


Figure 4. Supporting fault-tolerance.

key field to the data element. The index allows the subscribed heralds to distinguish between the various copies that they receive from a replicated application. Upon a read request, a herald first attempts to return a matching element having a fixed default index. When, after some appropriate time-out has expired, the requested element is still not available, a matching element with an index other than the default is returned. From that moment on it is assumed that the application corresponding to the default index has failed, and the subscription registration is updated accordingly. The index of the actually returned data element now becomes the new default.

A general overview of the distributed software architecture supporting fault-tolerance based on the various data and process replication techniques is given in Figure 4.

3.4. System Modifications and Extensions

In the embedded systems domain requirements on availability often make it necessary to support modifications and extensions while the current system remains online. There are two distinct cases to be considered.

- The upgrade is an extension to the system, introducing new applications and sorts but without further modifications to the existing system.

- The upgrade includes modification of existing applications.

Since the subscription registration is maintained dynamically by the heralds, it is obvious that the current protocol can deal with the first case without further refinements. After installing and starting a new application, it will automatically integrate.

The second case, clearly, is more difficult. One special, but important, category of modifications can be handled by a simple refinement of the heralds. Consider the problem of upgrading a system by replacing an existing application process with one that implements the same function, but using a better algorithm, leading to higher quality results. In many systems it is not possible to physically replace the current application with the new one, since this would require the system to be taken off-line.

By a refinement of the heralds it is possible to support on-line replacement of applications. If an application performs a write operation, its herald attaches an additional key field to the data element representing the application's version number. Upon a read request, a herald now first checks whether multiple versions of the requested instance are available in the local database. If this is the case, the instance having the highest version number is delivered to the application - assuming that higher numbers correspond to later releases. From that moment on, all data elements with lower version numbers, received from the same herald, are discarded. In this way an application can be dynamically upgraded, simply by starting the new version of the application, after which it will automatically integrate and replace the current version.

4. Conclusion

Due to the inherent complexity of the environment in which large-scale embedded systems operate, combined with the stringent requirements regarding temporal behaviour, availability, robustness, and maintainability, the design of these systems is an intricate task. Coordination models offer the potential of separating functional requirements from other aspects of system design. We have presented a software architecture for large-scale embedded systems that incorporates a separate coordination model. We have demonstrated how, starting from a relatively simple model based on a shared data space, the model can be successively refined to meet the requirements that are typical for this class of systems.

Over the past years SPLICE has been applied in the development of commercially available command-and-control, and traffic management systems. These systems consist of some 1000 applications running on close to 100 processors interconnected by a hybrid communication network. Experience with the development of these systems confirms that the software architecture, including all of the refinements discussed, significantly reduces the complexity of the design process [3]. Due to the high level of decoupling between processes, these systems are relatively easy to develop and integrate in an incremental way. Moreover, distribution of processes and data, fault-tolerant behaviour, graceful degradation, and dynamic reconfiguration are directly supported by the architecture.

References

[1] J.-P. Banatre, D. Le Metayer, "Programming by Multiset transformation", Communications of the ACM, Vol. 36, No. 1, 1993, pp. 98-111.

[2] M. Boasson, "Control Systems Software", IEEE Transactions on Automatic Control, Vol. 38, No. 7, 1993, pp. 1094-1107.

[3] M. Boasson, "Complexity may be our own fault", IEEE Software, March 1993.

[4] M. Boasson, Software Architecture special issue (guest editor), IEEE Software, November 1995.

[5] N. Carriero, D. Gelernter, "Linda in Context", Communications of the ACM, Vol. 32, No. 4, 1989, pp. 444-458.

[6] D. Garlan, D. Le Metayer (Eds.), “Coordination Languages and Models”, Lecture Notes in Computer Science 1282, Springer, 1997.

[7] D. Gelernter, N. Carriero, “Coordination Languages and their Significance”, Communications of the ACM, Vol. 35, No. 2, 1992, pp. 97-107.

[8] K. Jackson, M. Boasson, “The importance of good architectural style”, Proc. of the workshop of the IEEE TF on Engineering of Computer Based Systems, Tucson, 1995.

[9] G.-C. Roman, H.C. Cunningham, “Mixed Programming Metaphors in a Shared Dataspace Model of Concurrency”, IEEE Transactions of Software Engineering, Vol. 16, No. 12, 1990, pp.1361-1373.

2.2 – Component Architecture Framework – An Approach to the Enterprise Architecture Development in a Risk Environment

Tomas FEGLAR
International Consultant in Information Systems Research and Architecture
Vondrousova 1199, 163 00 Prague 6
CZECH REPUBLIC

feglar@centrum.cz

1.0 PRIMARY GOALS OF PROCESS INTEGRATION AND SYSTEMS ENGINEERING DISCIPLINE

Primary goals of process integration must directly support missions that are planned and owned by Business Planners and Owners:

- To assist Business Process (BP) Planners and Owners to understand, describe and continuously develop their business processes independently on application, IT infrastructure and physical environment.
- To protect BP Planners and Owners interests against risks by the way that guarantees continuous and measurable quality improvements.

The first primary goal corresponds to the Army Force management and Force Development (AFMP, FDP) structure which encapsulates a lot of hierarchically organized business processes.

The second primary goal corresponds to the Information Technology Landscape that supports AFMP/FDP structure and which is exposed to threats.

Systems Engineering (SE) received increased attention as ways are sought to reverse the trend of increasing project failure, particularly in large information systems which are exposed to internal and external disturbances.

SE engineering discipline distinguishes a lot of SE processes; four of them are the most critical for a development of systems supporting AFMP and FDP:

- Architectural Design Process (ADP);
- Risk Management Process (RMP);
- Information Management Process (IMP); and
- Security Management Process (SMP).

These processes are usually applied inconsistently and result that outcomes are available to BP Planners and Owners as stove pipe solutions. To overcome these serious limitations we have developed new approach – Component Architecture Framework (CAF) that integrates all four SE processes, support them with appropriate architecture frameworks and modeling methods.

2.0 ARCHITECTURE DESIGN PROCESS (ADP)

ADP combines TOGAF, Zachman and C4ISR frameworks. TOGAF is applied for architecture development; architectural products are stored in Zachman matrix that also includes mappings to the C4ISR views and products.

3.0 RISK MANAGEMENT PROCESS (RMP)

CAF associates outcomes of this process with a risk driven strategic concept that can be structured in accordance with enterprise strategy using profiles like ISO/IEC 17799, NATO Risk profile and others.

4.0 INFORMATION MANAGEMENT PROCESS (IMP)

CAF supports particular ITIL (Information Technology Infrastructure Library) modules, primarily Service Level Management, Availability Management, and Continuity Planning.

5.0 SECURITY MANAGEMENT PROCESS (SMP)

Risk driven strategic concept synthesized as a result of the RMP includes two large groups of countermeasures – security mechanisms and security operating procedures. SMP allows keeping a control over security mechanism implementation and over security responsibility (various stakeholders that have a set of security operating procedures included in their job description).

6.0 CONCLUSION

CAF approach includes also tools that support all four SE processes mentioned above with common graphical interface and integrated dictionary (CAF database). This approach was successfully applied in a development of Human Resource and TOE systems.

2.3 – High-Availability Solutions to Common Software Failures

Frédéric Michaud
Frédéric Painchaud
Defence Research and Development Canada – Valcartier
2459 Pie-XI Blvd North, Québec, QC, Canada, G3J 1X5

August 16, 2006

Abstract

The Canadian government, especially the Department of National Defence and the Canadian Forces, has a strong need for secure and reliable information systems. Currently used, mass-market systems tend to be very poor with respect to security and reliability, since they routinely contain serious bugs and vulnerabilities. While a redesign of these systems would be a sound long-term solution, an effective short-term solution is an imperative. Mature high-availability products for mass-market operating systems are now available and we believe they could effectively and robustly prevent the effects of some classes of failures. We propose to evaluate this hypothesis.

Introduction

The Canadian government, DND[1], and CF[2] are increasingly dependent on information systems, which need to offer a very high level of security, reliability, and fault-tolerance [1]. However, current information systems are inadequate for many reasons. First, many legacy systems currently in use were designed before the Internet was widespread and were not supposed to be exposed to a world-wide network. Second, they were built with technologies and programming languages that are prone to vulnerabilities and bugs that were not known at the time (e.g., buffer overflows). Since these systems are needed for a foreseeable future, special care should be taken to prevent the

[1] Department of National Defence
[2] Canadian Forces

exploitation of these vulnerabilities. Finally, other systems were built on top of mass-market operating systems and integrated with widely-available applications, which have a poor security and reliability record [3, 4, 5]. These mass-market components are routinely used in contexts that exceeds the level of security and reliability they were designed to offer [2].

We believe that the ideal solution would be a complete redesign of these systems with the use of adapted programming languages and frameworks, aimed at providing better software fault-tolerance, to get rid of the problems at their source. Indeed, better designs with more explicit security and reliability requirements and the use of safe programming languages and technologies not prone to vulnerabilities, such as Java or Ada, would be a very good start. However, systems with critical security and reliability requirements are rather expensive to specify, develop, procure, operate and maintain, because of the time and expertise involved. Fundamental research on subjects related to the survivability of systems is also needed to solve remaining problems and questions [6]. This ideal solution is therefore necessarily a long-term solution.

In the short term, something else must be done. The good news is that mature high-availability products for mass-market operating systems are now available. These products claim that they can “wrap” existing applications and run them in a virtualized environment, allowing their execution to continue even if a fatal error happens. However, we could not find a complete, independent report on their evaluation. Therefore, we need to investigate these products, and this is our proposition, in order to know how good these solutions are and which threats they can mitigate.

The following section presents the family of high-availability solutions that are of interest. Then, section 2 discusses information system threats and how some of them could be mitigated with high-availability solutions. Finally, section 3 details our evaluation’s goals and work plan.

1 High-Availability Solutions

High-availability solutions wrap the execution of an existing system and helps it attain a higher level of availability by mitigating the effects of hardware and software errors. This is mainly done by the use of redundancy, where a failed component is replaced with a working one so that the system can continue to offer its service.

Interesting solutions are those that wrap the entire system, not only a single component, as a RAID disk array does. These solutions, called *high-availability clusters* [8, 9], generally use clones of the entire system as redundant nodes.

High-availability clusters can work in many modes:

Monitor and Replace Without State Transfer The system is monitored for errors and when a fatal one happens, the failed node is shut down and replaced with a hot standby. As the state of the failed node is not transferred to the new one, the execution cannot continue and the system has to be reinitialized, including remote clients. Therefore, this type of high-availability cluster simply automates the reinitialization process.

Monitor and Replace With State Transfer Again, the system is monitored for errors and when a fatal one happens, the failed node is shut down and replaced with a hot standby. However, the state of the failed node is preserved and transferred to the new node in order to continue the execution as if the failure never happened. A small downtime can occur while the state is transferred from the failed node to the hot standby.

Mask Failures With Virtualization This time, the system runs inside a virtualized environment, made of many nodes that run the system in parallel. When a fatal error occurs in one node, it is simply masked by using the result of another node (or many others), without any downtime or reinitialization. This approach has less drawbacks than other ones, but it is much more complex to implement correctly.

In a nutshell, high-availability clusters can either restart a failed system or mask the failure so that the execution can continue as if nothing happened, sometimes minus a small downtime. Important questions emanate from these observations:

- How useful is restarting a system when a fatal error happens? Is the error going to occur again after a restart?
- Which errors can be masked and which cannot?
- Do applications need to be aware of the cluster or can everything be transparent?
- What kind of applications make the state transfer impractical?

These are some of the questions we would like to answer after our evaluation of high-availability clusters.

1.1 Products of Interest

A short research on the Internet revealed many interesting high-availability products that can be evaluated. Here is a non-exhaustive list with a description from their respective vendor:

Marathon everRun$^{\mathbf{FT}}$ *This product synchronizes two unmodified servers to create a virtual application environment that runs on both of them simultaneously. If one server fails, the other server enables the application to continue operating without interruption* [10].

Microsoft Clustering Services *This service provides high availability and scalability for mission-critical applications such as databases, messaging systems, and file and print services. Multiple servers (nodes) in a cluster remain in constant communication. If one of the nodes in a cluster becomes unavailable as a result of failure or maintenance, another node immediately begins providing service, a process known as failover. Users who are accessing the service continue to access the service, and are unaware that it is now being provided from a different server* [11].

Veritas Cluster Server *Veritas Cluster Server can detect faults in an application and all its dependent components, including the associated database, operating system, network, and storage resources. When a failure is detected, Cluster Server gracefully shuts down the application, restarts it on an available server, connects it to the appropriate storage device, and resumes normal operations* [12].

VMware Virtual Infrastructure *VMware High Availability provides easy to use, cost effective high availability for applications running in virtual machines. In the event of server failure, affected virtual machines are automatically restarted on other production servers with spare capacity* [13].

Linux High-Availability Project *It provides monitoring of cluster nodes, applications, and provides a sophisticated dependency model with a rule-based resource placement scheme. When faults occur, or a rule-change occurs, the user-supplied rules are then followed to provide the desired resource placement in the cluster* [14].

On paper, these products seem very promising. Obviously, our evaluation would include thorough testing of these products in order to validate these claims.

1.2 Added Value Provided by High-Availability Solutions

As a side note, it is important to realize that availability offered by these solutions can be leveraged in other contexts and can provide non-negligible added value, such as:

Non-Disruptive Maintenance Upgrades and common hardware and software maintenance can be performed while the system is running, without its users noticing any downtime. Furthermore, if the system's state is extensively logged (which should be the case most of the time), modifications that prove to be erroneous can be rollbacked.

Server Consolidation In order to achieve fault-tolerance, virtualization is often used in high-availability solutions and it can also be used to run many virtual servers on the same hardware. It is thus possible to consolidate many less-used servers on one powerful computer, simplifying management.

Advanced Monitoring & Logging State transfers and virtualization in high-availability clusters need extensive monitoring and logging which is ideal for good forensics. Therefore, if, for whatever reason, a system that uses a high-availability cluster is attacked and fails, the failure has the potential to be deeply analyzed from the logs.

2 Problems & Threats

This section looks at threats that current systems face everyday [7] and how a high-availability solution could help, by either restarting the failed component or by masking the effects of the fault. Restarting a failed component is a sound solution only if the cause of failure is *transient*, or temporary. If the cause of failure is still present after a restart, the component will fail again and will keep being restarted over and over again. Masking the effects of a fault may also not always be possible, if, for instance, all the duplicate nodes fail simultaneously and give the same erroneous output.

2.1 Environmental Faults

Environmental faults are failures that originate from outside the system itself, such as a power failure, a network connectivity loss (cut cable), natural disasters, etc. Since the problem lies outside the system, restarting it will not change much, unless the problem goes away while the system is restarting.

If the system cannot deal with a temporary outage of a specific kind (it cannot progress from a failed state to a correct state), restarting it may be necessary after the outage. Masking the fault would require a distributed system with geographically-distant nodes that cannot be subject to the same environmental faults.

We see a limited use of high-availability solutions for these kinds of faults.

2.2 Hardware Faults

Hardware faults occur when a hardware component of the system stops working correctly and reports an error. Examples include a crashed disk, failed parity checking while reading memory, burned power supply, etc.

High-availability solutions were first designed to handle hardware faults and are generally considered to work well in that context.

2.3 Software Faults

Software faults occur when the execution of a program diverges from "what it should be", i.e., its specification. For instance, a program could write data outside the bounds of one of its buffers. This leads to memory corruption and can be the cause of a crash. Errors in computation can also emerge if the program reads outside the bounds of one of its buffers. Another example of a program crash is if the program attempts a division by zero.

Software fault-tolerance is a vast domain being intensively researched right now. All problems and questions are not yet solved and answered. We want to start by investigating what currently-available high-availability solutions have to offer in handling software fault-tolerance of existing systems. The answer to "What can be done when a system crashes because of a software fault?" is not easy to find. However, we believe that high-availability solutions could be useful when a transient software error occurs.

2.4 Malicious Acts

Systems can also crash because of malicious acts like denial-of-service and code injection attacks. High-availability solutions cannot mitigate code injection attacks because if a system is vulnerable to code injection, making it more available will not solve the problem. However, systems wrapped by high-availability solutions can be more resilient to denial-of-service attacks because if the attack slows down one of the nodes of the system, load balancing could be used to redistribute resources and achieve better performance. If the attack crashes the system, it could be automatically restarted.

3 Goals & Work Plan

To sum up, our main goals for our evaluation are as follows:

1. Evaluate monitors' error-detection performance.
   (a) Which faults can be detected?
   (b) Which monitoring approach do they use?
2. Assess how useful is a restart and for which kind of faults.
3. Assess how useful and feasible is to mask faults.
4. Determine if high-availability solutions are transparent to the applications.
5. Determine the limits of each product in general.

And our draft work plan:

Summer and Fall 2006 : Feasibility study and state-of-the-art report.

Spring 2007 : Preparation (development of tests, setup, etc.).

Summer 2007 : Evaluation.

Fall 2007 : Final report.

References

[1] Martin Croxford, Roderick Chapman, *Correctness by Construction: A Manifesto for High-Integrity Software*, CrossTalk, December 2005.

[2] Gregory Slabodkin, *Software glitches leave Navy Smart Ship dead in the water*, Government Computer News, July 13, 1998.

[3] SANS Institute, *The SANS Top 20 Internet Security Vulnerabilities*, http://www.sans.org/top20/

[4] SecurityFOCUS, *BugTraq Mailing List Archive*, http://www.securityfocus.com/archive/1

[5] McAfee, *McAfee Threat Center – Security Vulnerabilities*, http://www.mcafee.com/us/threat_center/vulnerabilities.html

[6] Peter G. Neumann, *Practical Architectures for Survivable Systems and Networks*, Computer Science Laboratory, SRI International, June 2000.

[7] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr, *Basic Concepts and Taxonomy of Dependable and Secure Computing*, IEEE Transactions on Dependable and Secure Computing, Vol. 1, No. 1, January-March 2004 (23 pages).

[8] Gregory Pfister, *In Search of Clusters, 2nd Edition*, Prentice Hall, 1998, 608 pages.

[9] Evan Marcus, Hal Stern, *Blueprints for High Availability, 2nd Edition*, John Wiley & Sons, 2003, 624 pages.

[10] http://www.marathontechnologies.com/

[11] http://www.microsoft.com/windowsserver2003/technologies/clustering/default.mspx

[12] http://www.symantec.com/Products/enterprise?c=prodinfo&refId=20

[13] http://www.vmware.com/vinfrastructure/

[14] http://www.linux-ha.org/

2.4 – A Looming Fault Tolerance Software Crisis?

August 2, 2006

Alexander Romanovsky

Newcastle University

alexander.romanovsky@ncl.ac.uk

Abstract. Experience suggests that it is edifying to talk about software crises at NATO workshops. It is argued in this position paper that proper engineering of fault tolerance software has not been getting the attention it deserves. The paper outlines the difficulties in building fault tolerant systems and describes the challenges software fault tolerance is facing. The solution being advocated is to place a special emphasis on *fault tolerance software engineering* which would provide a set of methods, techniques, models and tools that would exactly fit application domains, fault assumptions and system requirements and support disciplined and rigorous fault tolerance throughout all phases of the life cycle. The paper finishes with an outline of some directions of work requiring special focused efforts from the R&D community.

1. Fault tolerance misuse

As reported by Flavio Cristian in the 80s [1], field experience with telephone switching systems showed that up to two thirds of system failures were due to design faults in exception handling or recovery algorithms.

Let us look into what is happening now.

- The Interim Report on Causes of the August 14th 2003 Blackout in the US and Canada [2] clearly shows that the problem was mostly caused by badly designed fault tolerance: poor diagnostics of faults, longer-than-estimated time for component recovery, failure to involve all necessary components in recovery, inconsistent system state after recovery, failures of alarm systems, etc.
- Tony Hoare reports [3] that in some MS systems more than 10% of code is dedicated to executable assertions. Yet as we all know, many customers are still unhappy with the quality of these products
- The authors of an ICSE 2006 paper [4] have experimentally found that in a 10 million LOC real-time embedded control system, misused exception handling introduced 2-3 bugs per 1 KLOS.
- Another paper, just accepted to IVNET 2006 [5], shows that in eight .NET assemblies (which represent application, library and infrastructure levels), over 90% of exceptions that the code can throw are not documented.
- Paper [6] by IBM researchers reports typical patterns of exception handling misuse and abuse in five customer and one proprietary J2EE applications, referring to them as “bad coding practice”. It was found, for example, that one in ten classes swallows exceptions without doing anything about them.

We keep making mistakes in designing fault tolerance! The situation is deteriorating as the complexity of software and systems in general is growing, causing an increase in the complexity of fault tolerance, as computer-based systems proliferate more widely in business, society and individuals' activities.

In spite of the fact that a plethora of fault tolerance mechanisms have been developed since the 70s, that there is a good understanding of the basic principles of building fault tolerant software and that a considerable fraction of requirements analysis, run time resources, development effort and code are now dedicated to fault tolerance, we might well be on our way to a fault tolerant software crisis. At present, fault tolerance is not trustworthy as it is the least understood, documented and tested part of the system, is frequently misused or poorly designed, regularly left until too late in the development process, not typically introduced in a systematic, disciplined or rigorous way, and often not suitable for the specific situations in which it is applied.

2. Fault tolerance: challenges and difficulties

Fault tolerance means can and will undermine overall system dependability if not applied properly. The following are some of the main challenges in the area:

- Fault tolerance means are *difficult* to develop or, when they are provided by some dedicated support, to use – this is because they increase system *complexity* by adding a new dimension to the reasoning about system behaviour. Their application requires a deep understanding of the intricate links between normal and abnormal behaviour and states of systems and components, as well as system state and behaviour during recovery.
- Fault tolerance (software diversity, rollback, exception handling) is *costly* as it always uses redundancy. Rather than improve fault tolerance, system developers far too often prefer to spend resources on extending functionality. We cannot and/or do not always want to put a cost on failures.
- System designers are *reluctant* to think about faults at the early phases of development. Fault tolerance is often considered to be an implementation issue. Moreover, fault tolerance is often “added” *after* the normal part of the system is developed, which makes it less effective, may require system redesign or result in faulty fault tolerance.
- There is a lack of appropriate training for, education about or good practice in, fault tolerance:
  - We do not really know what counts as a good fault tolerant program. We usually know well only how to write programs and components that assume (unjustifiably) that nothing will go wrong
  - Developers of many applications fail to apply even the basic principles of software fault tolerance. There is no focus on clearly defining fault assumptions from the very start, early error detection, recursive system structuring for error confinement, minimising and ensuring error confinement and error recovery areas, extending component specification with a concise and complete definition of failure modes, etc.
- It is imperative that fault tolerance means *fit* the system, the types of faults (i.e. the fault assumptions), the application domain, the development paradigm, the execution environment and the system characteristics. We need suitable *fault tolerance abstractions* for a variety of particular situations.

3. Fault assumptions and application fault tolerance

We believe that due to

- an increase in hardware quality and a reduction in hardware cost (e.g. hardware replication is cheap)

- a dramatic rise in software complexity and volume
- the involvement of new actors (non-professional users, multiple organisations, critical infrastructures)
- a growing complexity of the environment in which systems operate,

for many applications hardware faults are no longer the predominant threat. These applications include a wide range of safety-, life-, business- and money-critical systems – see, for example, recent studies by J.-C. Laprie [7], J. Knight [8] and by the Standish Group [9]. The *predominant types* of faults to be tolerated are

- application software faults (including design faults)
- environmental and infrastructural faults/deficiencies
- potentially damaging changes in systems, components, environments and infrastructures
- mismatches of components composed together (including mismatches of fault tolerance mechanisms [10])
- architectural and organisational mismatches and system-level inconsistencies
- degradation of services provided by components and systems
- organisational, human and socio-technical faults.

Such faults cannot be tolerated (and the system recovered) by hardware or middleware means alone, without involving application software. This is why we need to include fault tolerance measures into application system development (be it top-down or bottom-up or a mix of both).

4. Fault tolerance and software development

Fault tolerance needs to be engineered in a disciplined and rigorous way. In agreement with a number of my colleagues working in fault tolerance, I see the way forward in pursuing the following directions:

- integrating fault tolerance measures (diversity, exception handling, backward error recovery, etc.) into system models starting from the early architectural design
- making fault tolerance-related decisions for each appropriate model by modelling faults, fault tolerance measures and dedicated redundant resources. In particular, we need to focus on fault tolerant software architectures
- ensuring correct transformations of those models that enrich fault tolerance measures and make models more concrete and detailed
- making fault tolerance verification and validation part of system development
- developing dedicated tool support for fault tolerance development
- providing domain-specific application-level fault tolerance mechanisms and abstractions.

Clearly, there has been some research done in these areas. Yet if we look at the proceedings of some best conferences relevant to dependability and software engineering, such as ICSE, DSN, ESEC/FSE and EDCC, we will see that these topics are at best peripheral. It is my strong belief that more focused efforts are needed to achieve fault tolerance which neither fails nor requires fault tolerance itself.

5. Where to look for solutions

In this section I would like to briefly introduce some of the R&D directions which I believe are or will be contributing to the successful engineering of fault tolerance.

Architecting fault tolerant systems is now becoming an active research area. We need to focus on introducing specialised architectural solutions:

- supporting all main fault tolerance mechanisms (exception handling with error confinement, software diversity, atomic actions, etc.)
- introducing specific fault tolerance solutions (such as adaptors and protective wrappers for COTS component integration – [11])
- making existing and widely accepted architectures fault tolerant
- ensuring tolerance of architectural mismatches [12].

It is essential that fault tolerance is supported by a set of specialised *patterns and styles* that would assist developers at all steps of the life cycle. These should include specialised architectural, refinement, decomposition, design, implementation and model transformation patterns and styles.

Where appropriate, fault tolerance should be developed *formally* to ensure its “correctness by construction”. This needs to be supported by a development environment with a set of specialised tools. We should be able to model faults and fault tolerance, to express, prove and check specific fault tolerance properties of these models and to refine them by refining both fault assumptions and fault tolerance means.

Different faults and their tolerance need to be considered *at the appropriate phases* of the life cycle and further refined and decomposed during development. This needs to start with the requirement phase.

To avoid making software more complex and introducing new faults, the fault tolerance *mechanisms and abstractions* being developed should *fit* the types of faults, the application domain, the development paradigm, the execution environment and the system characteristics and requirements.

Fault tolerance and fault tolerant evolution. Both system evolution and dynamic upgrade should ensure the preservation or the controlled and predictable changes of system fault tolerance. It is systems going through online modifications that are mostly vulnerable to faults, so we need specialised fault tolerance mechanisms that will ensure dependable modifications.

Some of the recent and ongoing activities that are directly related to engineering fault tolerant systems:

- RODIN - Rigorous Open Development Environment for Complex Systems, FP6 IST STREP project (2004-2007)[1]
- FME 2005 Workshop on rigorous engineering of fault tolerant systems (REFT 2005, Newcastle, July 2005) and a follow-up State of the Art LNCS collection [13]
- Workshop on engineering fault tolerant systems in Luxemburg (EFTS 2006)[2]. June 2006
- Edited collection of papers on engineering fault tolerant systems to be published in 2007
- A series of workshops on architecting dependable systems (WADS at ICSE and DSN in 2002-2006)[3] and three follow-up LNCS collections [14-16].

Acknowledgements. My thanks go to many people with whom these ideas have been discussed, but first of all, to Brian Randell, Cliff Jones, Jörg Kienzle, Nicolas Guelfi and Fernando Castro Filho. This work is supported by the IST RODIN Project.

[1] http://rodin.cs.ncl.ac.uk/

[2] http://se2c.uni.lu/tiki/tiki-index.php?page=Efts2006Overview

[3] http://www.cs.kent.ac.uk/events/conf/2006/wads/

References

1. F. Cristian. Exception handling. In Dependability of Resilient Computers, T. Anderson (Ed.). Blackwell Scientific Publications, 1989. pp. 68-97.

2. Interim Report: Causes of the August 14th Blackout in the United States and Canada. Canada–U.S. Power System Outage Task Force. November 2003. http://www.nrcan-rncan.gc.ca/media/docs/reports_e.htm.

3. T. Hoare. Assertions in modern software engineering practice. Invited talk. COMPSAC 2002. Oxford, UK, 26-29 August 2002.

4. M. Bruntink, A. van Deursen, T. Tourwé. Discovering Faults in Idiom-Based Exception Handling. ICSE 2006. 20-28 May 2006. Shanghai. China. ACM Press. pp. 242-251.

5. P. Sacramento, B. Cabral, P. Marques. Unchecked exceptions: can the programmer be trusted to document exceptions? Accepted for the 2nd Int. Conf. on Innovative Views of .NET Technologies (IVNET 2006). 2006. Florianopolis, Brazil.

6. D. Reimer, H. Srinivasan. Analyzing exception usage in large java applications. In Proceedings of ECOOP 2003 Workshop on Exception Handling in Object-Oriented Systems, July 2003.

7. J.-C. Laprie. Dependability of software-based critical systems. In Dependable Network Computing. D. R. Avresky (Ed.). 1999.

8. J. Knight. Assured Reconfiguration: An Architectural Core For System Dependability. Invited talk. ICSE 2005 Workshop on Architecting Dependable Systems. St. Louis, Missouri, USA, 17 May 2005.

9. J. Johnson. The Other Side of Failure! DSN 2006 Industry Session. June 26. Philadelphia, USA. 2006.

10. A. Avizienis. Infrastructure-Based Design of Fault-Tolerant Systems. In the Electronic Proceedings of the IFIP Int. Workshop on Dependable Computing and Its Applications (DCIA 98) January 12 - 14, 1998, Johannesburg, South Africa.

11. T. Anderson, B. Randell, A. Romanovsky. Wrapping the future. In the Proceedings of the IFIP Congress Topical Sessions. Toulouse. France. 2004. pp. 165-174.

12. R. de Lemos, C. Gacek, A. Romanovsky. Architectural Mismatch Tolerance. In Architecting Dependable Systems. LNCS 2677, 2003. pp. 175-194.

13. M. Butler, C. Jones, A. Romanovsky, E. Troubitsyna (Eds). Rigorous development of complex fault tolerant system. LNCS 4157. 2006.

14. R. de Lemos, C. Gacek, A. Romanovsky (Eds). Architecting Dependable Systems. LNCS 2677. 2003.

15. R. de Lemos, C. Gacek, A. Romanovsky (Eds). Architecting Dependable Systems II. LNCS 3069, 2004.

16. R. de Lemos, C. Gacek, A. Romanovsky (Eds). Architecting Dependable Systems III. LNCS 3549, 2005.

2.5 – Strategies for Achieving Robustness in Coalitions of Systems

Mary Shaw
School of Computer Science
Carnegie Mellon University
mary.shaw@cs.cmu.edu

November 2006

“Robustness” is an overarching property of software systems that includes, to various viewers and to various extents, elements of correctness, reliability, fault-tolerance, performance, security, usability (without surprises), accuracy, and numerous other properties. Robustness is a form of dependability that focuses on resilience to failures.

Many aspects of dependability and robustness have been explored extensively in the context of individual components. Modern software systems, however, are composed from multiple components. Often these components have not been designed to operate together. Increasingly these components are legacy code or even applications that can operate alone as well as in concert. Further, the components may be data or services as well as code. The challenge of individual components lies in understanding and managing the code, but the major challenge of modern systems lies in understanding and managing the interactions among the components. Large-scale system integration encounters new sources of problems, such as architectural mismatch, cross-platform portability, and side effects of evolution of the computing infrastructure.

This new setting qualitatively changes the nature of the software development and integration process.

Classical software	***Modern systems***
Localized	Distributed
Independent	Interdependent
Insular	Vulnerable
Installations	Communities
Centrally-administered	User-managed
Software	Information resource
Systems	Coalitions

In this setting, “coalition” seems like a more suitable label than “system” for the interacting collection of information technology components.

A number of strategies offer complementary approaches for achieving robustness in this setting, as suggested by Figure 1.

There are two general ways to deal with the possibility of bad things happening: *Prevent* them from happening at all and detect problems and *react* to them as they occur. We approach the former through validation and the latter through remediation. Within each of these categories we can identify (at least) three interesting cases.

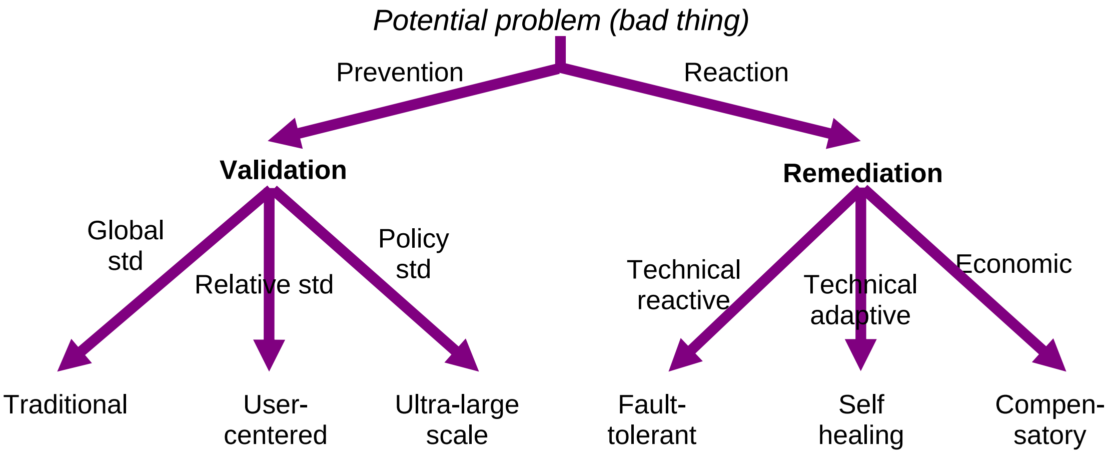


Figure 1: Approaches to robustness in modern software coalitions

Prevention

Prevention based on a global standard

This case is the focus of much of formal language theory and static program analysis, which attempt to make guarantees about programs based on the code of a system. In recent years this has focused on specific properties rather than complete specifications. The objective here is absolute guarantees.

This is also the focus of dynamic analysis including testing and dynamic analyses (e.g., of runtime behavior).

Prevention based on a relative standard

It is increasingly clear that the acceptability of a system to a specific user – and hence the robustness of the system in the eyes of that user – depends as much on the expectations of the user as it does on compliance with system specifications. Two issues arise.

First, the expectations of a given user may be either less demanding or more demanding than the system specification promises (or would promise if it existed). The first case is clear: the user might not use all of the precision, capability. or performance of the system or might be more tolerant of failures. The second case also arises, though: users often imagine what they hope the system may do and are unhappily surprised when it does not meet their expectations.

The second issue is a question of engineering cost-effectiveness: the cost of increasing robustness may not be justified by a user's actual needs. Rather, we need a way for individual users to determine whether a system is *sufficiently dependable* for their own needs.

Prevention based on a policy standard

We are beginning to come to grips with systems that are very large as measured by observable metrics such as lines of code, numbers of users, amount of data, and dependencies among components. But we continue to reason about them as if they were discrete systems subject to central control.

A more complex form of system is now emerging, with the Internet as a principal example. These systems are not simply larger versions of the systems we are familiar with. They feature

- Decentralized operation and control
- Conflicting, unknowable, diverse requirements
- Continuous evolution and deployment
- Heterogeneous, inconsistent, changing elements
- Indistinct people/system boundary
- Normal failures triggered by complex system coupling

These features preclude central design. These systems grow organically as a result of the actions of independent, possibly competitive users. They require new forms of acquisition and policy that are more akin to zoning laws – ways to govern independent evolution – than to conventional system specifications. [Software Engineering Institute. *Ultra Large-Scale Systems: The Software Challenge of the Future*. 2006 http://www.sei.cmu.edu/uls/]

Reaction

Reaction based on traditional reactive techniques

This case is the focus of classical fault tolerance, with roots in classical hardware fault tolerance with explicit set points or specifications of error states. In this case, robustness thresholds are set explicitly and crossing a threshold triggers remedial action. The strategy is to characterize the states of the system and the transitions between those states.

Reaction based on adaptive techniques

A difficulty with traditional reactive techniques is that they must invest specification effort in precisely defining the internal states or thresholds. Sometimes the robustness property of interest is appropriately treated as a threshold, but more often a system degrades gradually from dependable to undependable operation. Choosing an exact threshold requires making a choice of a single point in this gray area of decline.

An alternative is to base reaction on adaptive techniques that respond with low intensity to mild decline and with increasing intensity as the situation deteriorates, but that do so as a general reaction to conditions rather than as an explicit state change. Biological *homeostasis* offers tantalizing examples.

Robustness can also be improved by budgeting computing capability for reflection: maintaining a model of expected system behavior, monitoring system performance, and triggering adaptation. In effect, this replaces classical fixed setpoints with a more sophisticated basis for adaptation.

Reaction based on economic mechanisms

Sometimes systems fail and dynamic recovery is not possible. The world at large recognizes this as a risk management problem. One common way to manage such risks is to convert low probability, high impact events into high probability, low impact events. Insurance is a common example: risks of low probability, high cost events are pooled over a population that shares similar risks. Each member of the pool contributes a “premium” – a know payment that creates a fund that is subsequently disbursed to the few members of the pool who actually encounter the event.

Insurance-based risk management is common in software-intensive businesses, but it has received little attention at the system level. Creating an insurance model for software-intensive systems would require the ability to predict failure rates for the insured system, a way to attribute system failures to specific components, a way to evaluate the cost of a failure, and a way to create the risk-sharing pool.

Explicit risk management also offers an opportunity to make triage decisions – to assess system degradation and drop nonessential functions. This approach is used in provisioning certain types of service bureaus: an economic decision may provide to maintain less capacity than potential peak load, planning to drop (and pay penalties) some clients when load peaks in order to provide capacity for higher-priority clients.

3.1 – NATO Workshop Prague 2006

Maarten Boasson
Faculty of Science
University of Amsterdam
Kruislaan 404
1098 SM Amsterdam
NETHERLANDS

Email: boasson@science.uva.nl

This section was received as a PowerPoint presentation in PDF format.

Click here to view Presentation

3.2 – SaGE, an Exception Handling System for Message-Oriented Programming

Christophe Dony
Université de Montpellier
LIRMM
161 rue Ada
34392 Montpellier Cedex 5
FRANCE

Email: dony@lirmm.fr

This section was received as a PowerPoint presentation in PDF format.

Click here to view Presentation

3.3 – Service-Oriented Architecture (SOA) Robustness: The Road Ahead

Tomas Feglar
International Consultant in Information Systems Research and Architecture
Vondrousova 1199, 163 00 Prague 6
CZECH REPUBLIC

Email: feglar@centrum.cz

This section was received as a PowerPoint presentation in PDF format.

Click here to view Presentation

3.4 – Strategies for Achieving Dependability in Coalitions of Systems

Mary Shaw
A.J. Perlis Professor
Institute for Software Research International
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213-3891
USA

Email: mary.shaw@cmu.edu

This section was received as a PowerPoint presentation in PDF format.

Click here to view Presentation

3.5 – Closed-Loop Management Patterns

Joe Sventek
Professor of Communications Systems
Department of Computing Science
University of Glasgow
17 Lilybank Gardens
Glasgow, Scotland G12 8RZ
UNITED KINGDOM

Email: joe@dcs.gla.ac.uk

This section was received as a PowerPoint presentation in PDF format.

Click here to view Presentation

4.0 – Service-Oriented Architecture (SOA) Robustness: The Road Ahead

Tomas Feglar
International Consultant in Information Systems Research and Architecture
Vondrousova 1199, 163 00 Prague 6
CZECH REPUBLIC
Phone: +420 235 313 380, Fax: +420 235 313 380

Email: feglar@centrum.cz

ABSTRACT

Service-Oriented Architecture is a significant and integral part of the whole enterprise strategy. It must harmonize business process re-engineering with a power of enterprise technology infrastructure focusing Stability and Agility on the Enterprise Business Processes tier and Robustness on the SOA services tier. Robustness can be very effectively applied for SOA Enterprise Solutions by two ways; using enterprise System Engineering Support models for Short-Term solutions and using Robustness Patterns at the lower layers of SOA architecture for Long-Term solutions. Because SOA is more managerial then technological problem we propose Robustness based SOA Roadmap.

Keywords – SOA architecture, availability, robustness, risk analysis and management, system engineering

1.0 INTRODUCTION

The armed forces in East European Countries are under massive redesigning that requires a combination of process oriented and technologically oriented efforts. The border between these two efforts stimulates research activities oriented to service delivery in accordance with the needs of enterprise agility. At the same time we can observe increasing popularity of modeling that combines three main components – People, Process, and Technology [13,12,2,5]. Process re-engineering results mutual influences of these three components but it has to be synchronized with information security planning that corresponds with the robustness as a mechanism improving system stability and security. This is very important indicium but we feel that it has to be encapsulated into more comprehensive system engineering discipline [14,15,4].

SOA is much matter of management as it is technology [17]. To understand all key SOA management issues is difficult. Nowadays more typical SOA management practices stress only some of these issues increasing “stove pipe” risks in SOA project management. It is especially critical in military domain where stabile services directly influence Force Management.

This paper attacks SOA Robustness strategy in the context of the SOA based Robustness Roadmap that is understood as one of possible ways how to improve SOA management applying system engineering approach.

SOA Roadmap includes three main phases (Figure 1):

- Phase 1: Synthesize Enterprise Application Integration (EAI) Environmental Model and Decision Support Models.
- Phase 2: Establish SOA based Scenario Landscape.
- Phase 3: Establish SOA Robustness Enterprise Solutions using System Engineering Support Models and Robustness Patterns.

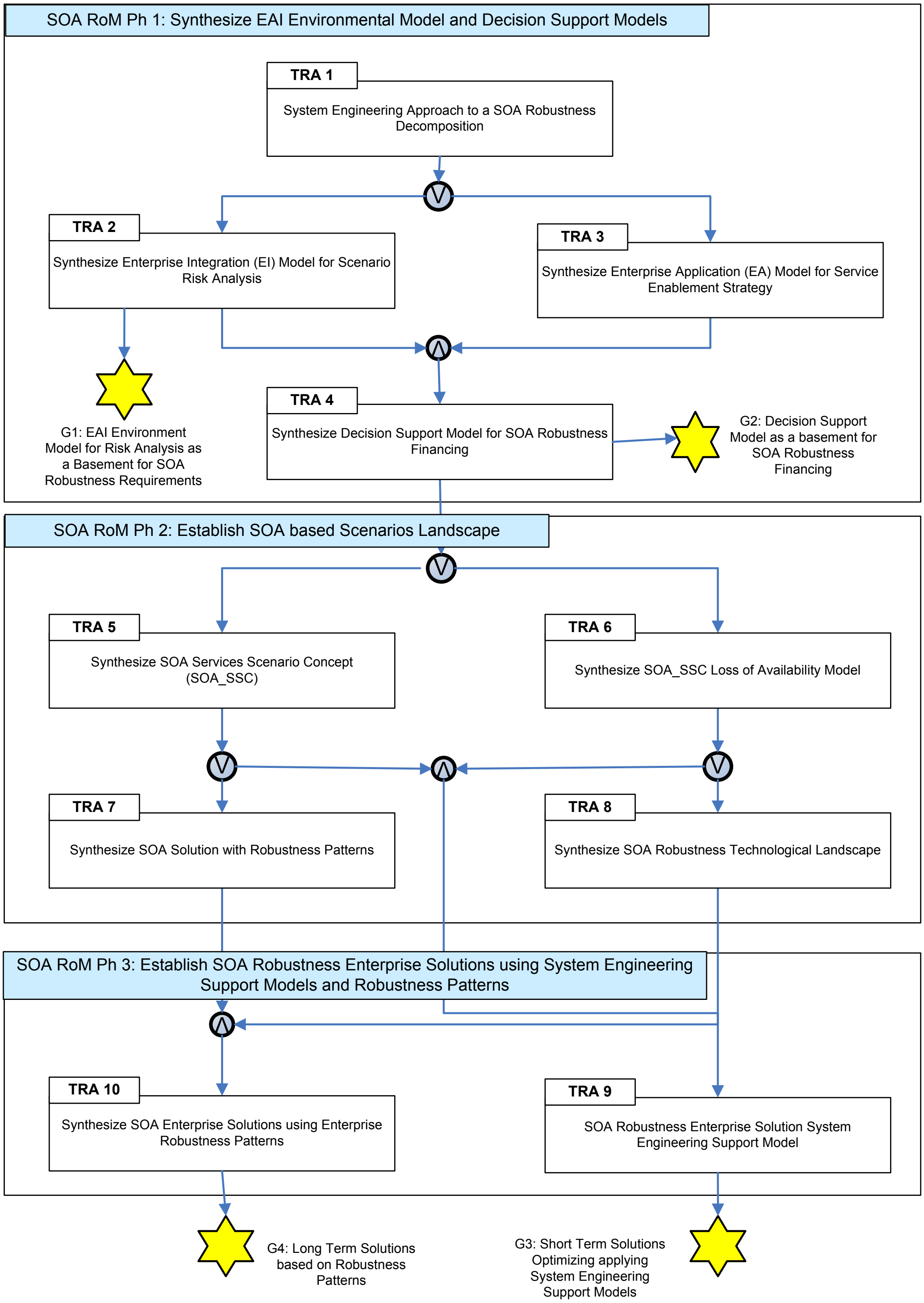

SOA RoM Ph 1: Synthesize EAI Environmental Model and Decision Support Models
TRA 1
System Engineering Approach to a SOA Robustness Decomposition
V
TRA 2
Synthesize Enterprise Integration (EI) Model for Scenario Risk Analysis
TRA 3
Synthesize Enterprise Application (EA) Model for Service Enablement Strategy
Λ
G1: EAI Environment Model for Risk Analysis as a Basement for SOA Robustness Requirements
TRA 4
Synthesize Decision Support Model for SOA Robustness Financing
G2: Decision Support Model as a basement for SOA Robustness Financing
SOA RoM Ph 2: Establish SOA based Scenarios Landscape
V
TRA 5
Synthesize SOA Services Scenario Concept (SOA_SSC)
TRA 6
Synthesize SOA_SSC Loss of Availability Model
V
Λ
V
TRA 7
Synthesize SOA Solution with Robustness Patterns
TRA 8
Synthesize SOA Robustness Technological Landscape
SOA RoM Ph 3: Establish SOA Robustness Enterprise Solutions using System Engineering Support Models and Robustness Patterns
Λ
TRA 10
Synthesize SOA Enterprise Solutions using Enterprise Robustness Patterns
TRA 9
SOA Robustness Enterprise Solution System Engineering Support Model
G4: Long Term Solutions based on Robustness Patterns
G3: Short Term Solutions Optimizing applying System Engineering Support Models

Figure 1: Robustness Based SOA Roadmap.

Following this roadmap we hope to achieve four management goals:

- Goal 1: EAI Environment Model for Risk Analysis as a Basement for SOA Robustness Requirements Specification.
- Goal 2: Decision Support Model as a basement for SOA Robustness Financing.
- Goal 3: Short-Term Solutions Optimizing applying System Engineering Support Models.
- Goal 4: Long-Term Solutions based on Robustness Patterns.

Each of these phases encapsulates relatively comprehensive modeling constructs; it was the reason why we decided to prepare this paper as the outcome of the model named "SOA Robustness – The Road Ahead" developed in accordance with Component Architecture Framework (CAF) approach [6,7,8]. The paper itself describes the most important milestones. Appendix 1 explains steps characterizing particular milestones.

Paragraph 1 describes Phase 1 that includes 4 tracks. The track TRA1 explains SOA Robustness decomposing this topic applying system engineering approach in accordance with ISO / IEC 15288 [15]. The track TRA2 explains Enterprise Integration (EI) Model synthesis for Scenario Risk Analysis. TRA4 is the most critical for SOA Robustness initiative financing. Using decision support modeling we developed some useful decision support templates that avoid decision makers intuitive, not optimal decision making.

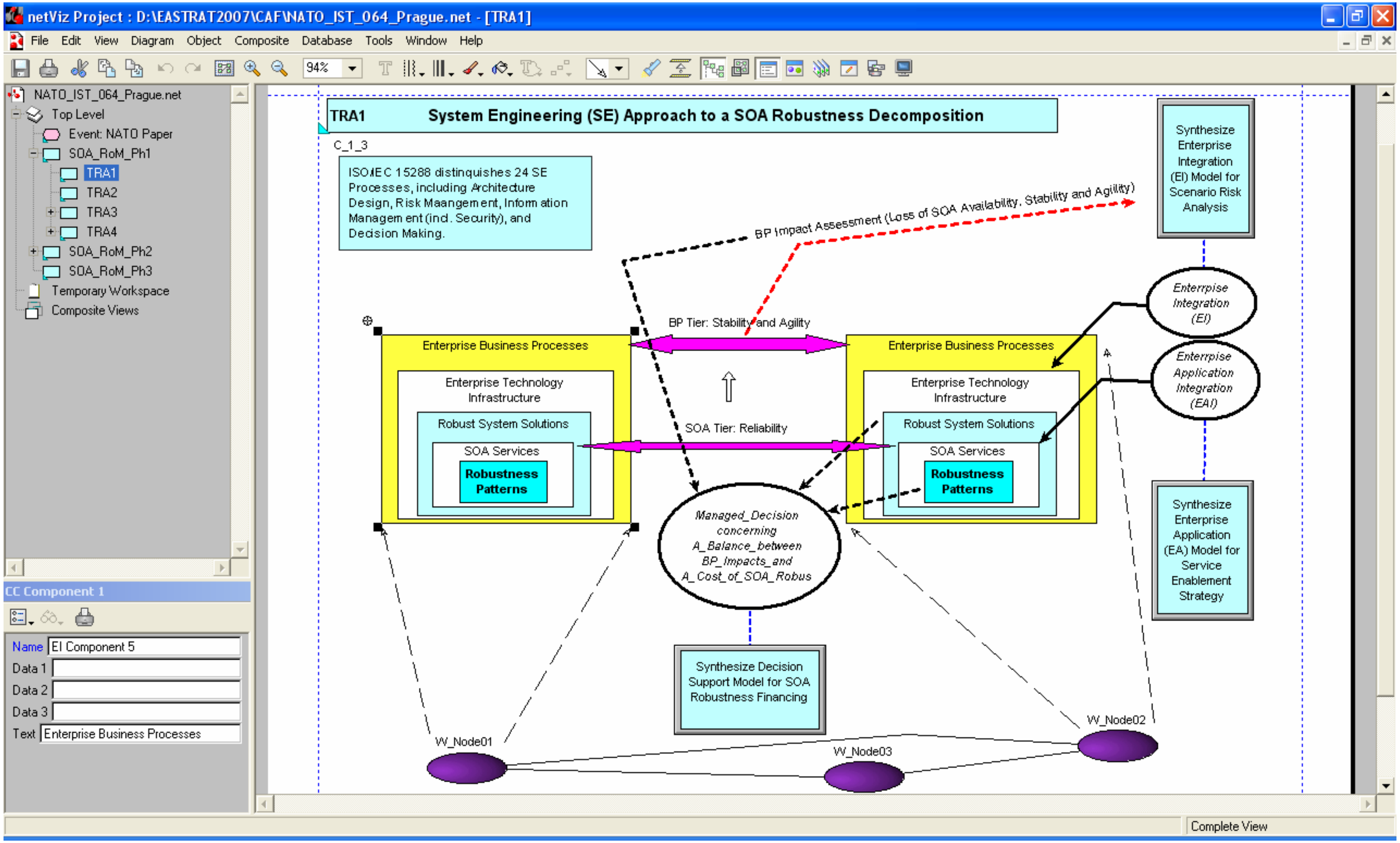
Second SOA Roadmap phase (paragraph 2) introduces two concepts that allow better management of huge amount of potential scenarios that relate to the SOA implementation. TRA5 illustrates key steps in designing SOA Services Scenario Concept. This Concept cam be further used for synthesis of SOA solution with robustness patterns (TRA7) or in a combination with Loss of Availability Model (TRA 6) for Short-Term SOA robustness solutions. TRA6 explains key items characterizing SOA Loss of availability Model that we need for SOA Robustness Landscape synthesis (TRA 8) and for optimizing of the Short Term system solutions.

Third SOA Roadmap phase (paragraph 3) consists of two tracks. TRA9 produces one of the main goals of SOA Roadmap – Short-Term Solutions Optimizing applying System Engineering Support Models. TRA10 is more implementation oriented and produces Long-Term Solutions based on Robustness Patterns.

2.0 ENTERPRISE APPLICATION INTEGRATION (EAI) ENVIRONMENT MODEL AND DECISION SUPPORT MODEL

The SOA Robustness decomposition approach considers that in the final stage of SOA deployment strategy Robustness Patterns will become a core of Enterprise Technology Infrastructure on which depend all key business processes (Figure 2). SOA based interoperability among Enterprises will require Stability and Agility at the business process (BP) tier and Reliability at the SOA services tier. To achieve these two very fundamental requirements we must be able to manage all important factors that significantly influence these requirements. In accordance with the Figure 2 they are:

- System Engineering experience based on Architecture Design Process, Risk Management Process, Information Management Process, and Decision Making Process.
- System Integration experience based on well understanding of Enterprise Technology Infrastructure.
- Enterprise Application experience based on well understanding of Service-Oriented Architecture and its practices.
- Decision Making experience capable combining all previously mentioned experiences with the robustness oriented goals that deal with a Balance between Business Process Impacts and a Cost of SOA Robustness.



netViz Project : D:\EASTRAT2007\CAF\NATO_IST_064_Prague.net - [TRA1]
File Edit View Diagram Object Composite Database Tools Window Help
94%
NATO_IST_064_Prague.net
Top Level
Event: NATO Paper
SOA_RoM_Ph1
TRA1
TRA2
TRA3
TRA4
SOA_RoM_Ph2
SOA_RoM_Ph3
Temporary Workspace
Composite Views
CC Component 1
Complete View
C_1_3
BP Tier: Stability and Agility
SOA Tier: Reliability
Enterprise Business Processes
Enterprise Technology Infrastructure
Robust System Solutions
SOA Services
Robustness Patterns
Enterprise Business Processes
Enterprise Technology Infrastructure
Robust System Solutions
SOA Services
Robustness Patterns
Enterrpise Integration (EI)
Enterrpise Application Integration (EAI)
W_Node01
W_Node02
W_Node03

Name: EI Component 5

Data 1:

Data 2:

Data 3:

Text: Enterprise Business Processes

TRA1 System Engineering (SE) Approach to a SOA Robustness Decomposition

ISO/IEC 15288 distinquishes 24 SE Processes, including Architecture Design, Risk Maangement, Information Management (incl. Security), and Decision Making.

BP Impact Assessment (Loss of SOA Availability, Stability and Agility)

Synthesize Enterprise Integration (EI) Model for Scenario Risk Analysis

Managed_Decision concerning A_Balance_between BP_Impacts_and A_Cost_of_SOA_Robus

Synthesize Decision Support Model for SOA Robustness Financing

Synthesize Enterprise Application (EA) Model for Service Enablement Strategy

Figure 2: System Engineering Approach to a SOA Robustness Decomposition.

Rapid increasing of the IT service delivery market stimulates research activities in the area of Service Quality and Marketing [19, 22]. Significantly less progress we can see in the area of service delivery in a risk environment [9]. It is serious problem; without risk motivated basement it is difficult to create robustness solution because we loss opportunity to argue robustness cost comparing it with business process impacts.

Figure 3 focuses Enterprise Integration (EI) Model; we consider this model as a model of environment in which SOA exists (SOA is primarily about applications). Enterprise node anatomy is analyzed from two perspectives – business process (BP) Impact and Threats / Vulnerabilities. Risk landscape results of these two perspectives. Particular risks can be calculating using automated tools like CRAMM [3] or analyzed through Threat Agents [16]. Next step following risk Analysis is a Risk Treatment. Figure 3 associates this step with designing of measures allowing decreasing risks to the acceptable level. We consider this framework as appropriate for merging with robustness oriented activities as for example Frederics' High-Availability Solutions [18].

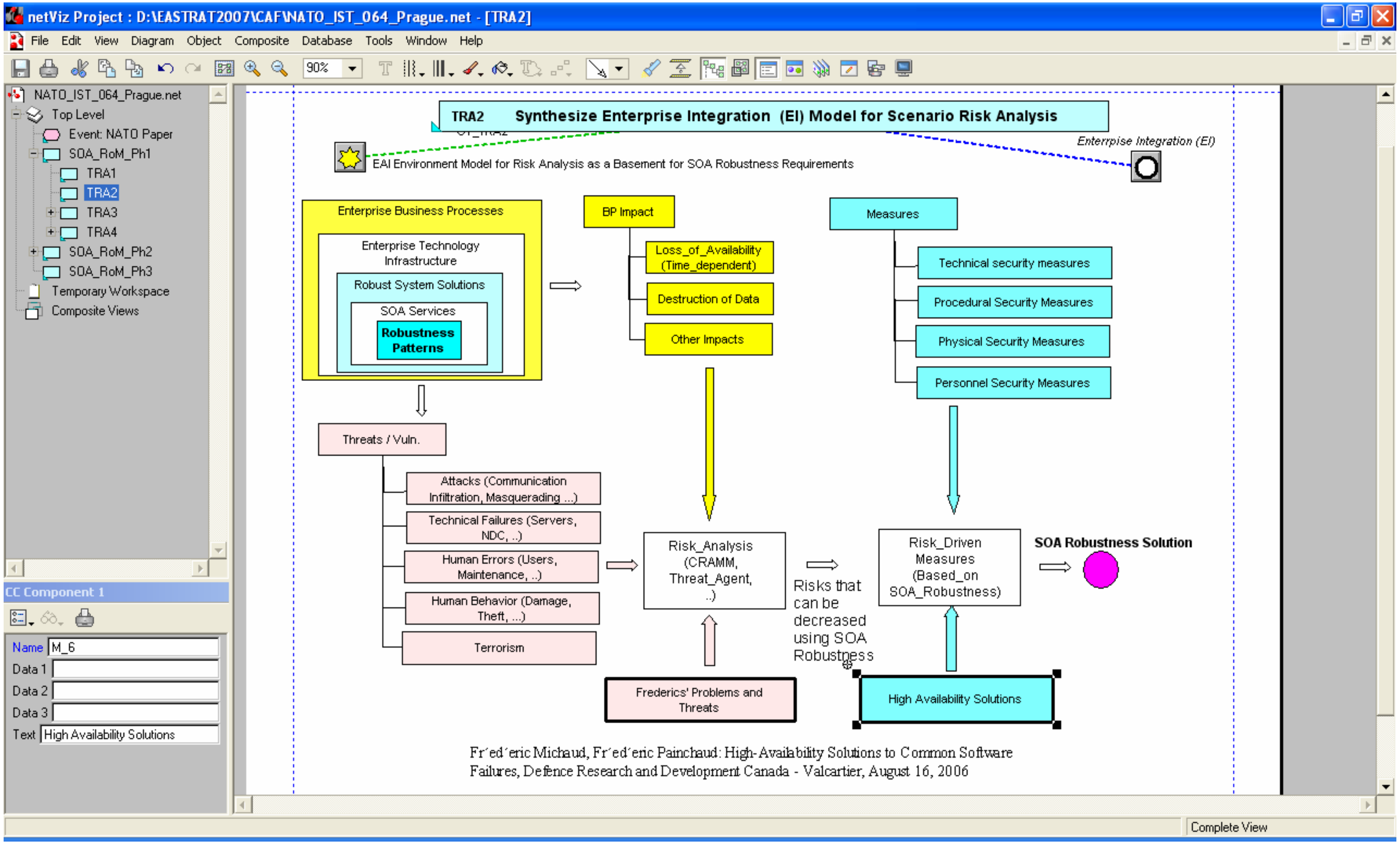
Service Enablement Strategy requires a development of Enterprise Application (EA) model that consists of three parts:

- Enterprise Integration Roadmap encapsulating SOA Architecture (Figure 4);
- Hitchin's Model of System Engineering, Defense Force horizons and SOA life cycle (Figure 5); and
- Force Management and Force Development Process and SOA Milestones (Figure 6 and 7).

Service Integration Architecture allows effective management of the SOA deployment strategy only if we understand influences of other architectures within particular enterprise (Figure 4). For example, Business Process Architecture allows us understanding a Vision (SOA deployment target) and Current Integration Assessment let us realistically assess constrains we must consider for our SOA milestones establishment.

SOA Concept distinguishes five components that differ from viewpoint of their life cycle (Figure 5). Robustness can be applied primarily for Technology Infrastructure (it's life cycle (LC) longs approximately 20 years) and for Services (their LC longs approximately 15 years). At the beginning of our paper we stress that SOA addresses a space between processes and technology respectively between process owners and IT specialists. To analyze People behavior in the SOA deployment strategy we need additional perspective oriented to the capability. It is the reason why we recommend combining SOA life cycle with Hitchin's model [13]. To achieve the Socio – Economic level, the Capability acquired during technology acquisition (Layer 2 of Hitchin's model) is not enough and must be followed with process owners oriented capability development (levels 3 and 4).

Enterprise SOA strategy distinguishes five milestones [8] in accordance with Figure 6 and 7. M1 relates to enterprise in which all key areas (Planning and Budgeting, HR Management, and Logistic) are supported by monolithic applications. For better synchronization of these milestones with East European armed forces transformation process we can omit first two milestones and start with milestone M3 and M4. The first one is typical for armed forces developing their key information systems as bespoke applications; milestone M4 is more appropriate for armed forces starting with ERP systems like SAP, Oracle Business Suite, PeopleSoft or Axapta.


netViz Project : D:\EASTRAT2007\CAF\NATO_IST_064_Prague.net - [TRA2]
File Edit View Diagram Object Composite Database Tools Window Help
90%
NATO_IST_064_Prague.net
Top Level
Event: NATO Paper
SOA_RoM_Ph1
TRA1
TRA2
TRA3
TRA4
SOA_RoM_Ph2
SOA_RoM_Ph3
Temporary Workspace
Composite Views
CC Component 1
TRA2 Synthesize Enterprise Integration (EI) Model for Scenario Risk Analysis
Enterprise Integration (EI)
EAI Environment Model for Risk Analysis as a Basement for SOA Robustness Requirements
Enterprise Business Processes
Enterprise Technology Infrastructure
Robust System Solutions
SOA Services
Robustness Patterns
BP Impact
Loss_of_Availability (Time_dependent)
Destruction of Data
Other Impacts
Measures
Technical security measures
Procedural Security Measures
Physical Security Measures
Personnel Security Measures
Threats / Vuln.
Attacks (Communication Infiltration, Masquerading ...)
Technical Failures (Servers, NDC, ..)
Human Errors (Users, Maintenance, ..)
Human Behavior (Damage, Theft, ...)
Terrorism
Risk_Analysis (CRAMM, Threat_Agent, ..)
Risks that can be decreased using SOA Robustness
Risk_Driven Measures (Based_on SOA_Robustness)
SOA Robustness Solution
Frederics' Problems and Threats
High Availability Solutions
Complete View

Name M_6

Data 1

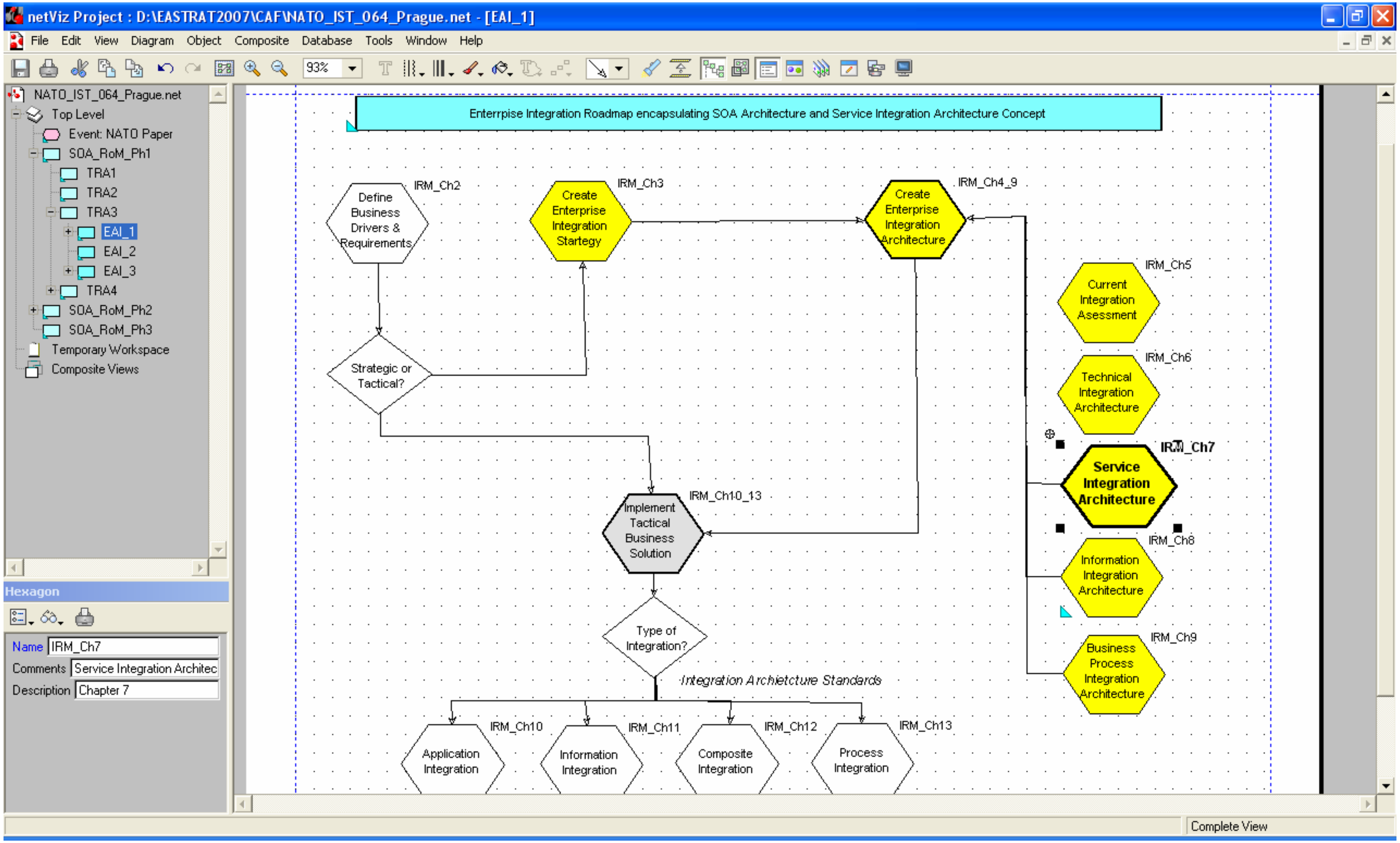
Data 2

Data 3

Text High Availability Solutions

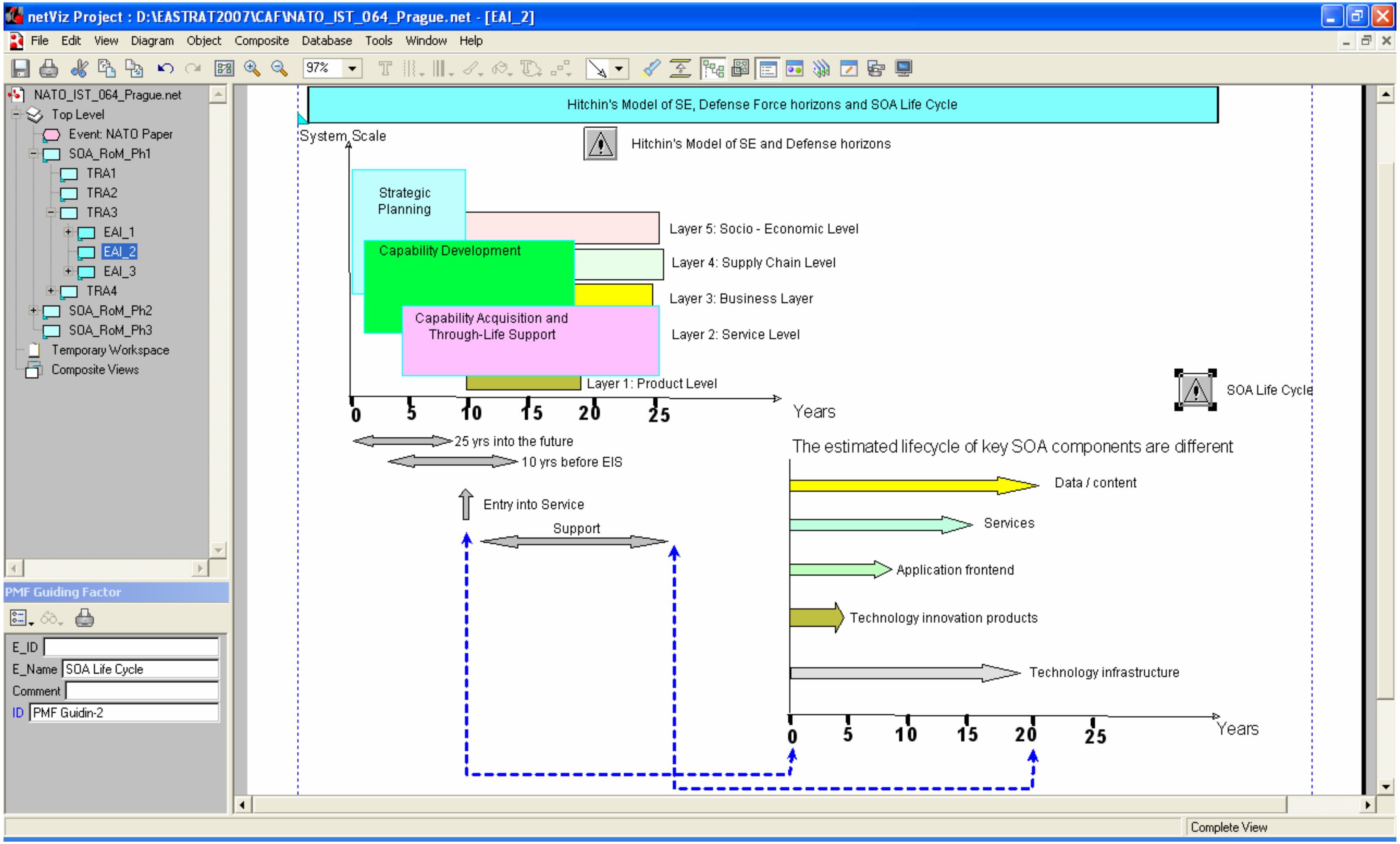
Fr´ed´eric Michaud, Fr´ed´eric Painchaud: High-Availability Solutions to Common Software Failures, Defence Research and Development Canada - Valcartier, August 16, 2006

Figure 3: Enterprise Integration (EI) Model for Scenario Risk Analysis.


netViz Project : D:\EASTRAT2007\CAF\NATO_IST_064_Prague.net - [EAI_1]
File
Edit
View
Diagram
Object
Composite
Database
Tools
Window
Help
93%
NATO_IST_064_Prague.net
Top Level
Event: NATO Paper
SOA_RoM_Ph1
TRA1
TRA2
TRA3
EAI_1
EAI_2
EAI_3
TRA4
SOA_RoM_Ph2
SOA_RoM_Ph3
Temporary Workspace
Composite Views
Hexagon
Enterrpise Integration Roadmap encapsulating SOA Architecture and Service Integration Architecture Concept
IRM_Ch2
Define Business Drivers & Requirements
IRM_Ch3
Create Enterprise Integration Startegy
IRM_Ch4_9
Create Enterprise Integration Architecture
IRM_Ch5
Current Integration Assessment
IRM_Ch6
Technical Integration Architecture
IRM_Ch7
Service Integration Architecture
IRM_Ch8
Information Integration Architecture
IRM_Ch9
Business Process Integration Architecture
Strategic or Tactical?
IRM_Ch10_13
Implement Tactical Business Solution
Type of Integration?
Integration Archietcture Standards
IRM_Ch10
Application Integration
IRM_Ch11
Information Integration
IRM_Ch12
Composite Integration
IRM_Ch13
Process Integration
Complete View

Name IRM_Ch7
Comments Service Integration Architec
Description Chapter 7

Figure 4: Service Integration Architecture in the Context of Enterprise Integration Strategy.



netViz Project : D:\EASTRAT2007\CAF\NATO_IST_064_Prague.net - [EAI_2]
File Edit View Diagram Object Composite Database Tools Window Help
97%
NATO_IST_064_Prague.net
Top Level
Event: NATO Paper
SOA_RoM_Ph1
TRA1
TRA2
TRA3
EAI_1
EAI_2
EAI_3
TRA4
SOA_RoM_Ph2
SOA_RoM_Ph3
Temporary Workspace
Composite Views
PMF Guiding Factor
Hitchin's Model of SE, Defense Force horizons and SOA Life Cycle
System Scale
Hitchin's Model of SE and Defense horizons
Strategic Planning
Layer 5: Socio - Economic Level
Capability Development
Layer 4: Supply Chain Level
Layer 3: Business Layer
Capability Acquisition and Through-Life Support
Layer 2: Service Level
Layer 1: Product Level
0 5 10 15 20 25
Years
25 yrs into the future
10 yrs before EIS
Entry into Service
Support
SOA Life Cycle
The estimated lifecycle of key SOA components are different
Data / content
Services
Application frontend
Technology innovation products
Technology infrastructure
0 5 10 15 20 25
Years
Complete View

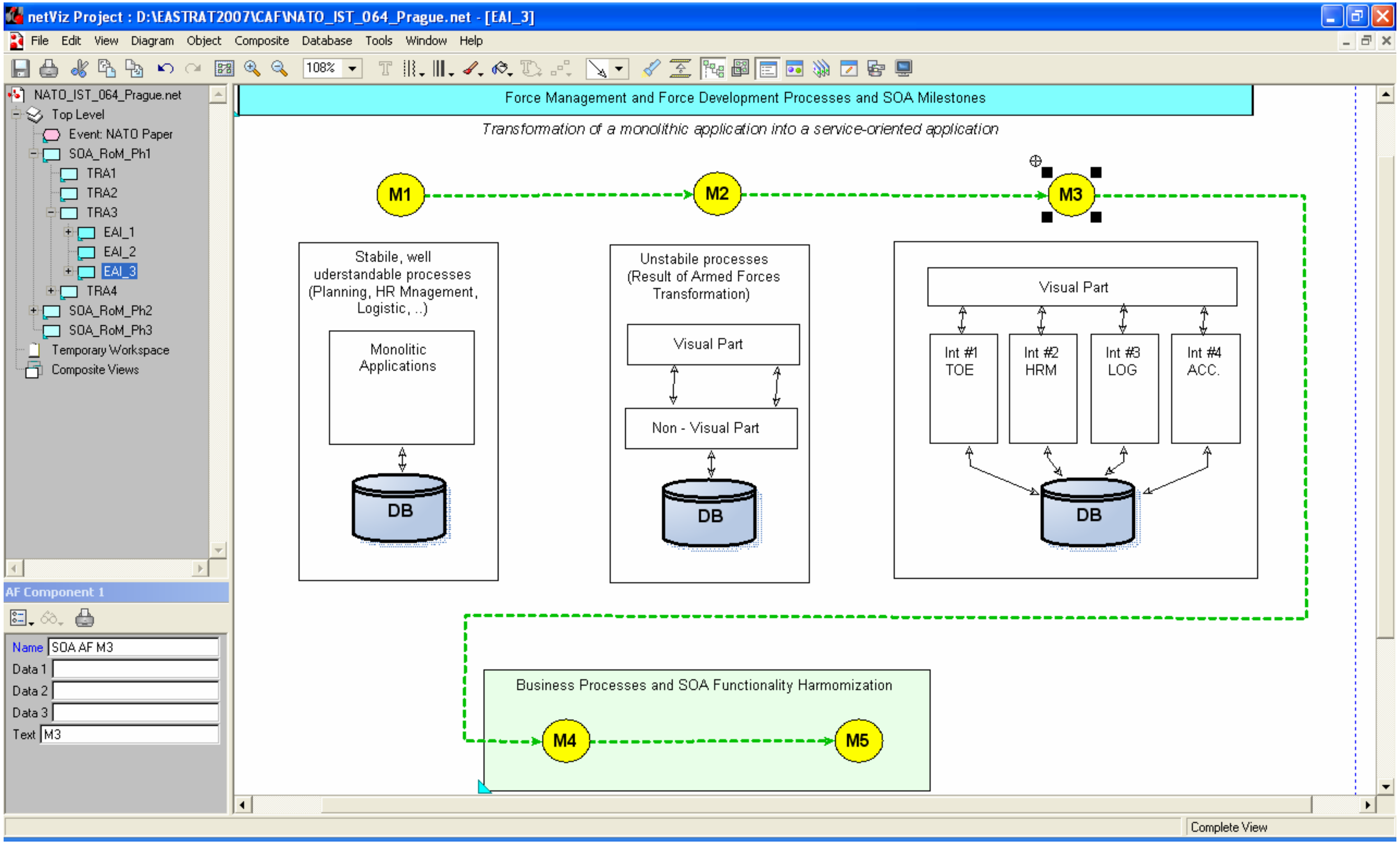
E_ID

E_Name SOA Life Cycle

Comment

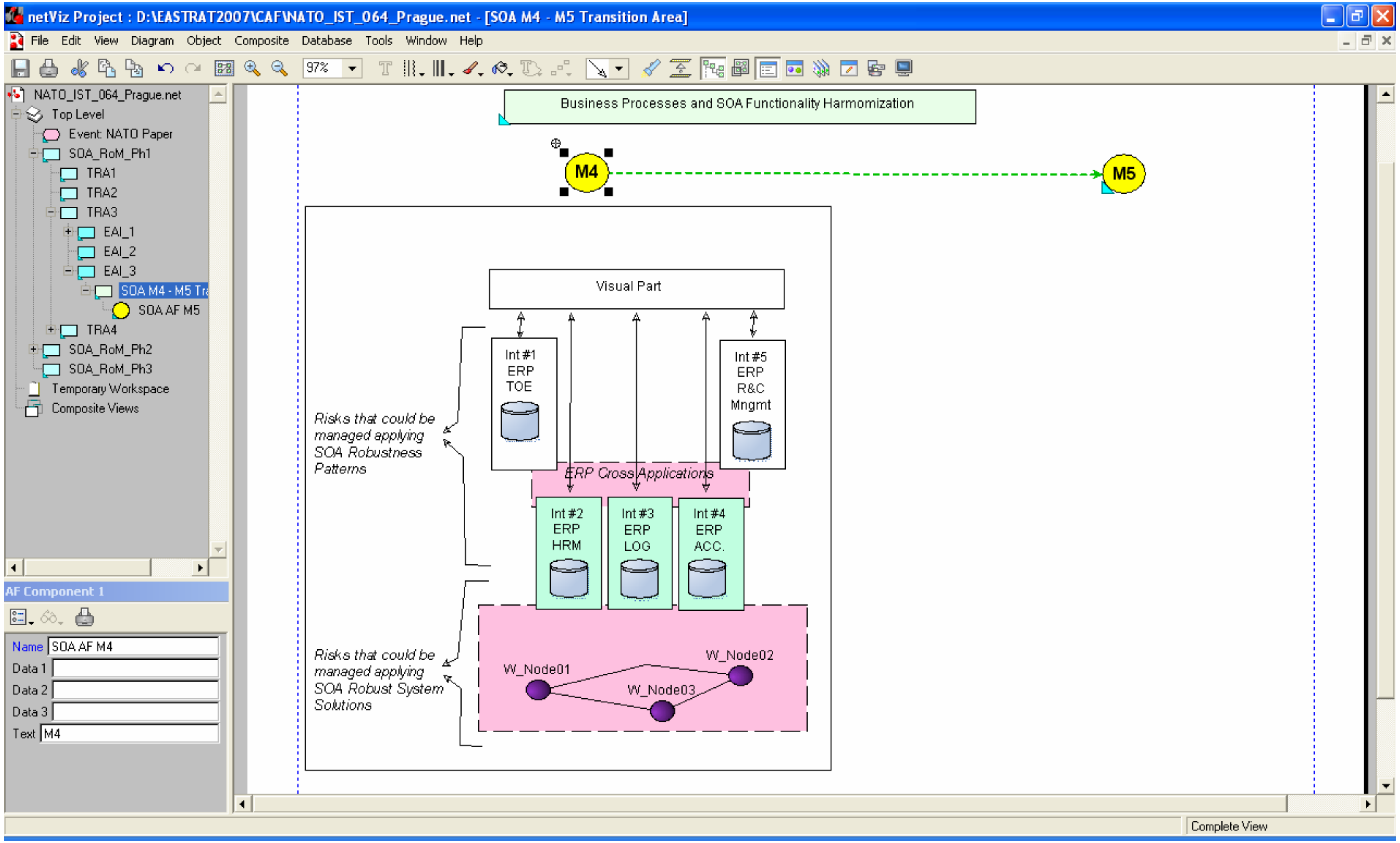
ID PMF Guidin-2

Figure 5: Harmonization of the SOA Life Cycle and Defence Force Capability Development.


netViz Project : D:\EASTRAT2007\CAF\NATO_IST_064_Prague.net - [EAI_3]
File Edit View Diagram Object Composite Database Tools Window Help
108%
NATO_IST_064_Prague.net
Top Level
Event: NATO Paper
SOA_RoM_Ph1
TRA1
TRA2
TRA3
EAI_1
EAI_2
EAI_3
TRA4
SOA_RoM_Ph2
SOA_RoM_Ph3
Temporary Workspace
Composite Views
AF Component 1
Force Management and Force Development Processes and SOA Milestones
Transformation of a monolithic application into a service-oriented application
M1
M2
M3
Stabile, well uderstandable processes (Planning, HR Mnagement, Logistic, ..)
Monolitic Applications
DB
Unstabile processes (Result of Armed Forces Transformation)
Visual Part
Non - Visual Part
DB
Visual Part
Int #1 TOE
Int #2 HRM
Int #3 LOG
Int #4 ACC.
DB
Business Processes and SOA Functionality Harmonization
M4
M5
Complete View

Name	SOA AF M3
Data 1	
Data 2	
Data 3	
Text	M3

Figure 6: Force Development and Force Management Processes and SOA Milestones.



Name SOA AF M4

Data 1

Data 2

Data 3

Text M4

Figure 7: Business Processes and SOA Functionality Harmonization (Milestone M4).

During explanation of SOA Robustness decomposition concept (Figure 2) we stress how important are Stability and Agility at the business process (BP) tier and Reliability at the SOA services tier. Achieving these two requirements at the end of the SOA Roadmap we should probably apply different approaches for milestone M3 and M4. **M3** seems to be more suitable for **long term robustness solutions** because it offers a possibility to develop SOA services using robustness patterns (in this case our SOA concept is deployed from Bottom to Up). **M4** requires Top Down approach. We have not enough time diving into technological aspects because our main attention must be given to process re-engineering. In this case **short term robustness** solutions seem to be more appropriate because they directly increase stability of ICT environment of business processes that are not stabile as result of transformation process.

Tracks TRA1 up to TRA3 clearly show a complexity that must be managed to successfully achieve SOA strategic goals. Robustness is very important piece of the whole picture but requires appropriate support at the decision makers' level.

It is worth to stress that right decision requires also modeling support that allows decision makers clear understanding the goal of decision, alternatives and criteria. Track TRA4 (Figure 8) describes decision oriented modeling for SOA Robustness Financing putting together.

Main goals:

- Goal 1: Justifying a Budget for Robustness oriented System Engineering Support in the information and communication technology (ICT) total cost of ownership (TCO) Context.
- Goal 2: Justifying a Budget for Robustness oriented System Engineering Support in the Information Security Context.
- Prioritize Robustness SOA Solutions Alternatives.

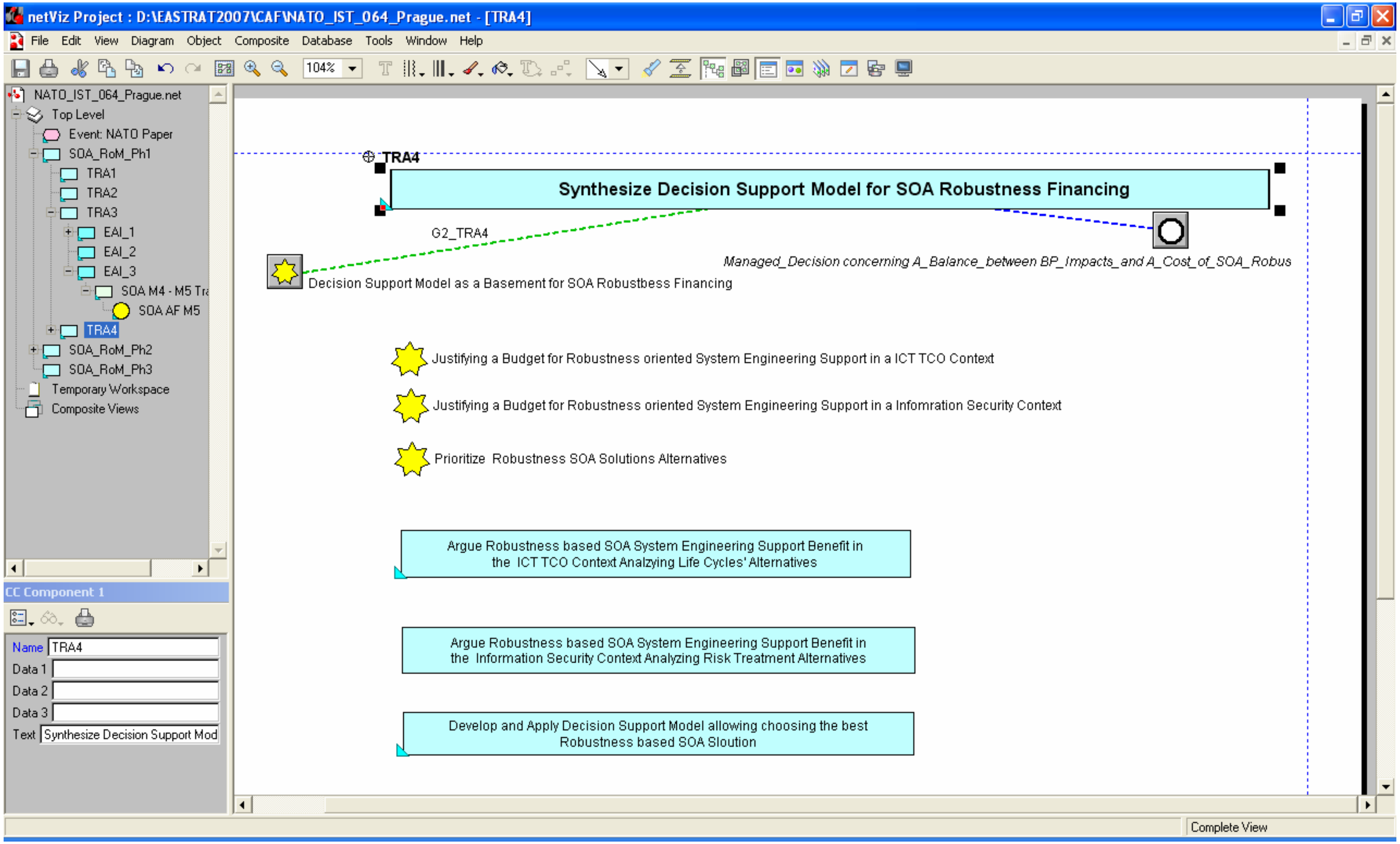
And main groups of activities:

- Arguing Robustness based SOA System Engineering Support Benefit in the ICT TCO Context analyzing SOA Life Cycle alternatives.
- Arguing Robustness based SOA System Engineering Support Benefit in the Information Security Context analyzing Risk Treatment Alternatives.
- Development and application of the Decision Support Model allowing choosing the best Robustness based SOA Solution.

Figures 9 and 10 briefly illustrate first and third activities.

One of the most popular ERP systems is SAP. A.W. Scheer – The main SAP architect – explains his experience with application of ARIS for SAP life cycle [21] (Figure 9). Scheer's experience relates two curves 1 and 2. Curve 1 characterizes total cost of ownership (TCO) across ERP life cycle when we omit system engineering support completely. Curve 2 introduces significant TCO savings especially during operational stage when we need constantly improved processes. Both curves also had shown unstable stages that follow ERP infrastructure upgrades. Robustness is the right mechanism to solve problems like these, but it must become an integral part of the whole ERP strategy from the beginning (see curve 3). TCO / ERP LC diagram in the Figure 9 is also acceptable for senior staff that can see the space for its decision.

Figure 10 illustrates a decision support modeling inspired by Frederics' paper that is also presented on this meeting [18]. They describe five products allowing significantly improve system robustness. We developed this model in accordance with AHP theory [20] applying EC 2000 software. More detailed explanation is included in Appendix 1.



Name TRA4
Data 1
Data 2
Data 3
Text Synthesize Decision Support Mod

Synthesize Decision Support Model for SOA Robustness Financing

Managed_Decision concerning A_Balance_between BP_Impacts_and A_Cost_of_SOA_Robus

Decision Support Model as a Basement for SOA Robustbess Financing

Justifying a Budget for Robustness oriented System Engineering Support in a ICT TCO Context

Justifying a Budget for Robustness oriented System Engineering Support in a Infomration Security Context

Prioritize Robustness SOA Solutions Alternatives

Argue Robustness based SOA System Engineering Support Benefit in the ICT TCO Context Analzying Life Cycles' Alternatives

Argue Robustness based SOA System Engineering Support Benefit in the Information Security Context Analyzing Risk Treatment Alternatives

Develop and Apply Decision Support Model allowing choosing the best Robustness based SOA Sloution

Figure 8: Decision Support Model for SOA Robustness Financing.

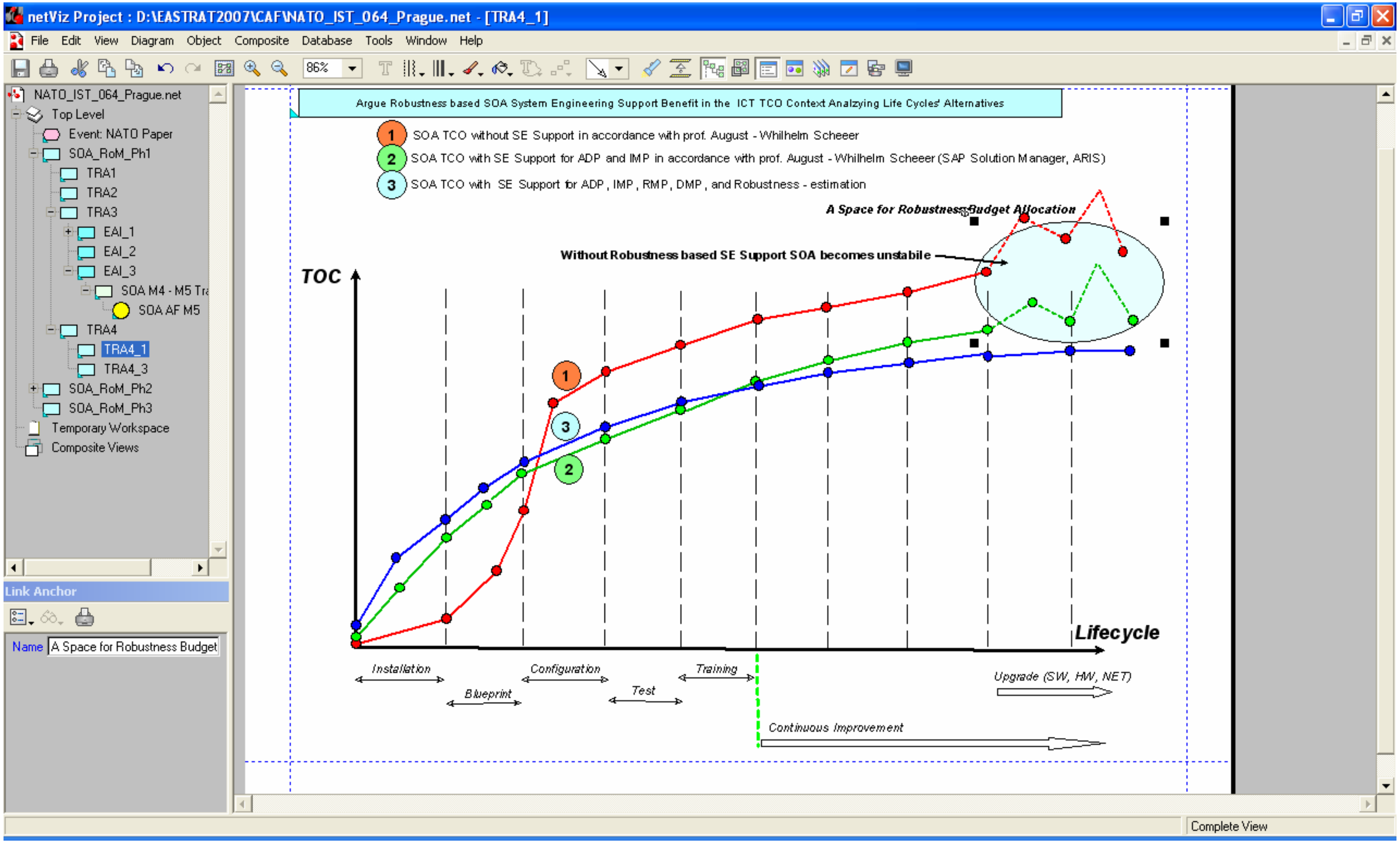
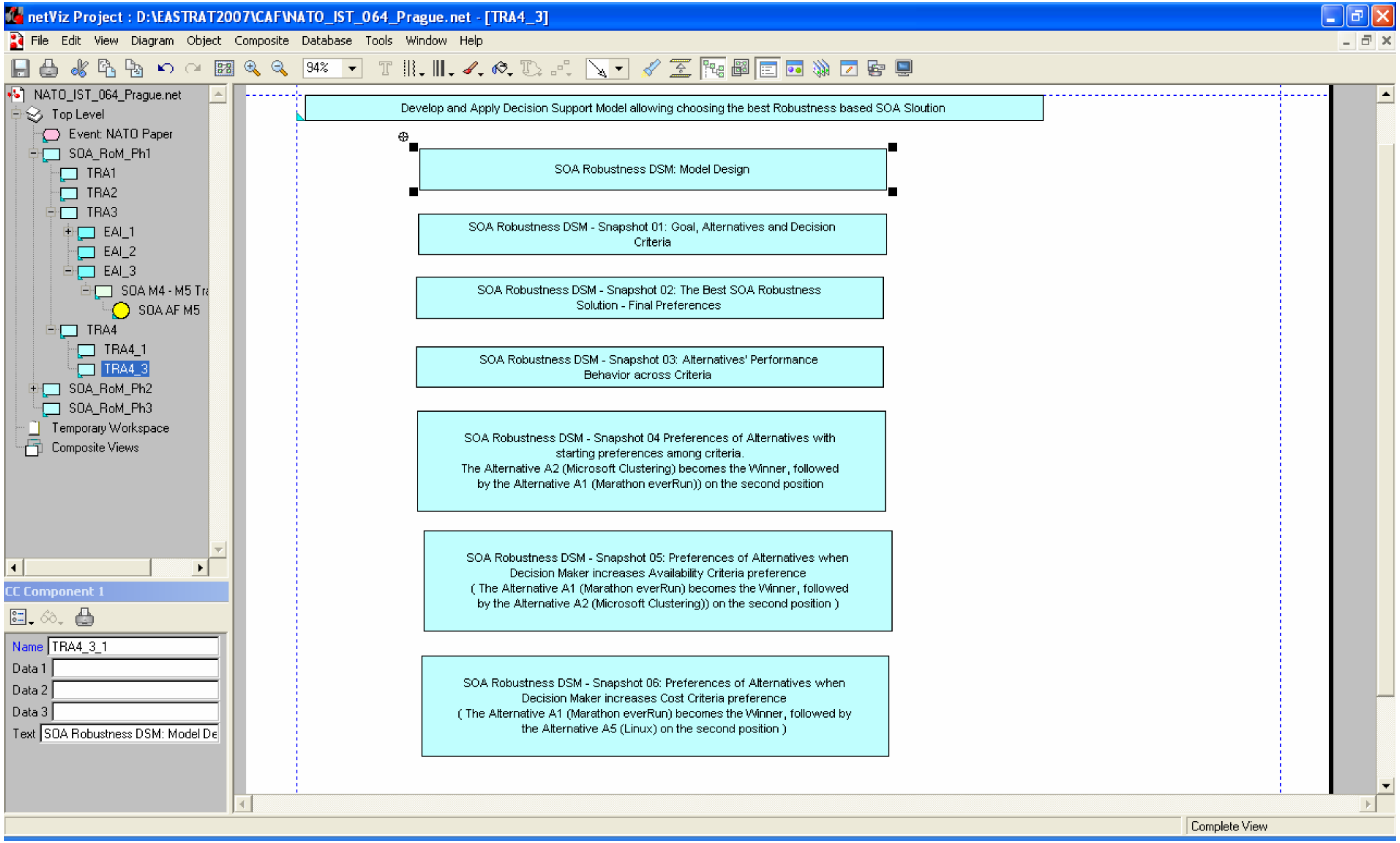

netViz Project : D:\EASTRAT2007\CAF\NATO_IST_064_Prague.net - [TRA4_1]
File Edit View Diagram Object Composite Database Tools Window Help
86%
NATO_IST_064_Prague.net
Top Level
Event: NATO Paper
SOA_RoM_Ph1
TRA1
TRA2
TRA3
EAI_1
EAI_2
EAI_3
SOA M4 - M5 Tra
SOA AF M5
TRA4
TRA4_1
TRA4_3
SOA_RoM_Ph2
SOA_RoM_Ph3
Temporary Workspace
Composite Views
Link Anchor
Name A Space for Robustness Budget
Argue Robustness based SOA System Engineering Support Benefit in the ICT TCO Context Analzying Life Cycles' Alternatives
1 SOA TCO without SE Support in accordance with prof. August - Whilhelm Scheeer
2 SOA TCO with SE Support for ADP and IMP in accordance with prof. August - Whilhelm Scheeer (SAP Solution Manager, ARIS)
3 SOA TCO with SE Support for ADP, IMP, RMP, DMP, and Robustness - estimation
A Space for Robustness Budget Allocation
Without Robustness based SE Support SOA becomes unstabile
TOC
1
3
2
Lifecycle
Installation
Blueprint
Configuration
Test
Training
Upgrade (SW, HW, NET)
Continuous Improvement
Complete View

Figure 9: Identification of the Space for Robustness Budget Allocation.



netViz Project : D:\EASTRAT2007\CAF\NATO_IST_064_Prague.net - [TRA4_3]

Name TRA4_3_1

Data 1

Data 2

Data 3

Text SOA Robustness DSM: Model De

Develop and Apply Decision Support Model allowing choosing the best Robustness based SOA Slоution

SOA Robustness DSM: Model Design

SOA Robustness DSM - Snapshot 01: Goal, Alternatives and Decision Criteria

SOA Robustness DSM - Snapshot 02: The Best SOA Robustness Solution - Final Preferences

SOA Robustness DSM - Snapshot 03: Alternatives' Performance Behavior across Criteria

SOA Robustness DSM - Snapshot 04 Preferences of Alternatives with starting preferences among criteria.
The Alternative A2 (Microsoft Clustering) becomes the Winner, followed by the Alternative A1 (Marathon everRun)) on the second position

SOA Robustness DSM - Snapshot 05: Preferences of Alternatives when Decision Maker increases Availability Criteria preference
(The Alternative A1 (Marathon everRun) becomes the Winner, followed by the Alternative A2 (Microsoft Clustering)) on the second position)

SOA Robustness DSM - Snapshot 06: Preferences of Alternatives when Decision Maker increases Cost Criteria preference
(The Alternative A1 (Marathon everRun) becomes the Winner, followed by the Alternative A5 (Linux) on the second position)

Figure 10: SOA Robustness Decision Support Model.

3.0 SERVICE-ORIENTED ARCHITECTURE (SOA) BASED SCENARIOS LANDSCAPE

First 4 tracks described in the previous paragraph are oriented primarily to the frameworks within which we can successfully manage SOA based Robustness strategy. But we also need bricks allowing us building the walls. SOA strategy offers a plenty of ways how to assembly SOA services to meet process requirements. Before we start doing this we must carefully arrange our workspace, another words we need scenarios.

Track TRA5 (Figure 11) shows:

- Enterprise SOA Services and Layers; and
- SOA Services Scenario Concept (SOA SSC).

SOA SCC distinguishes four basic parts necessary for process modeling in SOA environment. We explain these parts more detailed in Appendix 2 (see Material Request Order (MRO) process sample). SOA SSC uses following constructs:

- *Organizing Diagrams* let us understanding parts of an enterprise affected by a process.
- *Enterprise Service Structure Diagrams* let us understanding requested functionality decomposition across Enterprise SOA services.
- *Event Process Chain (EPC) Diagrams* let us visualize processes by the way that can be easily understand by business process owners.
- *Service Interaction Diagrams* are preferred by designers. These diagrams are usually derived from functional blocks used in EPC diagrams.

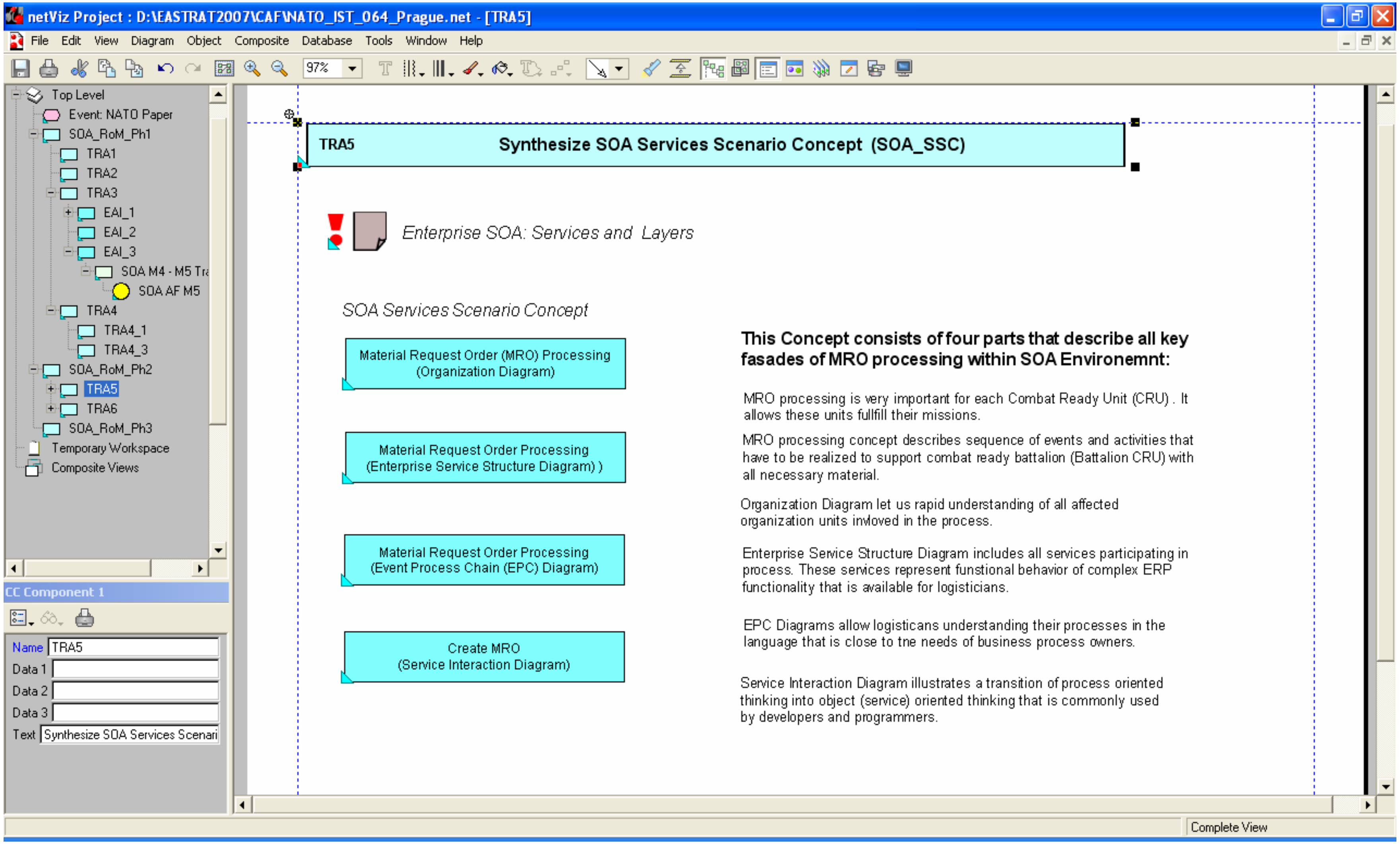
Figure 12 depicts SOA services and layers. Basic and Intermediary Services seems to be suitable for implementation of SOA Robustness Patterns that can be used for SOA Enterprise Solution building.

Robustness that applies high-availability products [18] requires a development of appropriate SOA SSC that can be used for synthesizing of a Loss of Availability Model. This kind of synthesizing starts with a decomposition of particular high level process (like operation planning (OPLAN)) into functionalities that can be overlapped by SOA services functionality (Figure 13). Figure 13 depicts a decomposition of a Material Order Request (MRO) processing at the Enterprise Level (this process is owned by logisticians) Process layer splits enterprise level process into three sub-processes – Material Management (MM), Logistic Execution, and Sales and Distribution (SD). Each of these sub-processes needs support of lower SOA layers. SOA SSC Loss of Availability Model usually consists of three modeling constructs (Figure 14). A content of these layers is depicted in Appendix 2.

4.0 SOA ROBUSTNESS ENTERPRISE SOLUTIONS USING SYSTEM ENGINEERING SUPPORT MODELS AND ROBUSTNESS PATTERNS

Last two tracks of the Robustness based SOA Roadmap (TRA9, TRA10) produce outcomes corresponding two main goals (Figure 15):

- Short Term Solutions Optimizing applying System Engineering Support Models (goal G3); and
- Long Term Solutions based on Robustness Patterns (goal G4).



Name TRA5
Data 1
Data 2
Data 3
Text Synthesize SOA Services Scenari

TRA5 Synthesize SOA Services Scenario Concept (SOA_SSC)

Enterprise SOA: Services and Layers

SOA Services Scenario Concept

Material Request Order (MRO) Processing (Organization Diagram)

Material Request Order Processing (Enterprise Service Structure Diagram))

Material Request Order Processing (Event Process Chain (EPC) Diagram)

Create MRO (Service Interaction Diagram)

This Concept consists of four parts that describe all key fasades of MRO processing within SOA Environemnt:

MRO processing is very important for each Combat Ready Unit (CRU) . It allows these units fullfill their missions.

MRO processing concept describes sequence of events and activities that have to be realized to support combat ready battalion (Battalion CRU) with all necessary material.

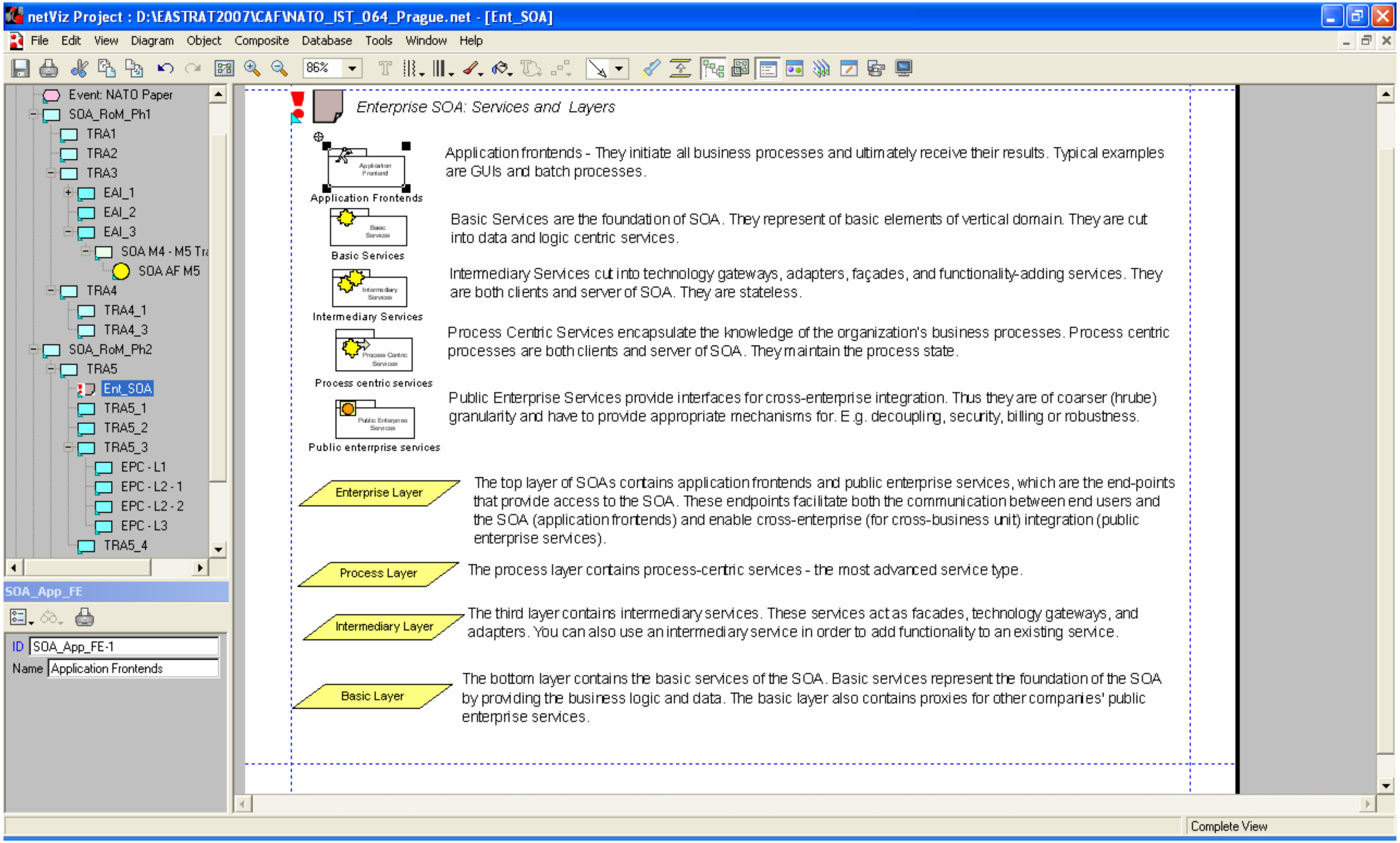
Organization Diagram let us rapid understanding of all affected organization units invloved in the process.

Enterprise Service Structure Diagram includes all services participating in process. These services represent funstional behavior of complex ERP functionality that is available for logisticians.

EPC Diagrams allow logisticans understanding their processes in the language that is close to tne needs of business process owners.

Service Interaction Diagram illustrates a transition of process oriented thinking into object (service) oriented thinking that is commonly used by developers and programmers.

Figure 11: SOA Services Scenario Concept, SOA Services and Layers.



ID SOA_App_FE-1
Name Application Frontends

Enterprise SOA: Services and Layers

Application frontends - They initiate all business processes and ultimately receive their results. Typical examples are GUIs and batch processes.

Basic Services are the foundation of SOA. They represent of basic elements of vertical domain. They are cut into data and logic centric services.

Intermediary Services cut into technology gateways, adapters, façades, and functionality-adding services. They are both clients and server of SOA. They are stateless.

Process Centric Services encapsulate the knowledge of the organization's business processes. Process centric processes are both clients and server of SOA. They maintain the process state.

Public Enterprise Services provide interfaces for cross-enterprise integration. Thus they are of coarser (hrube) granularity and have to provide appropriate mechanisms for. E.g. decoupling, security, billing or robustness.

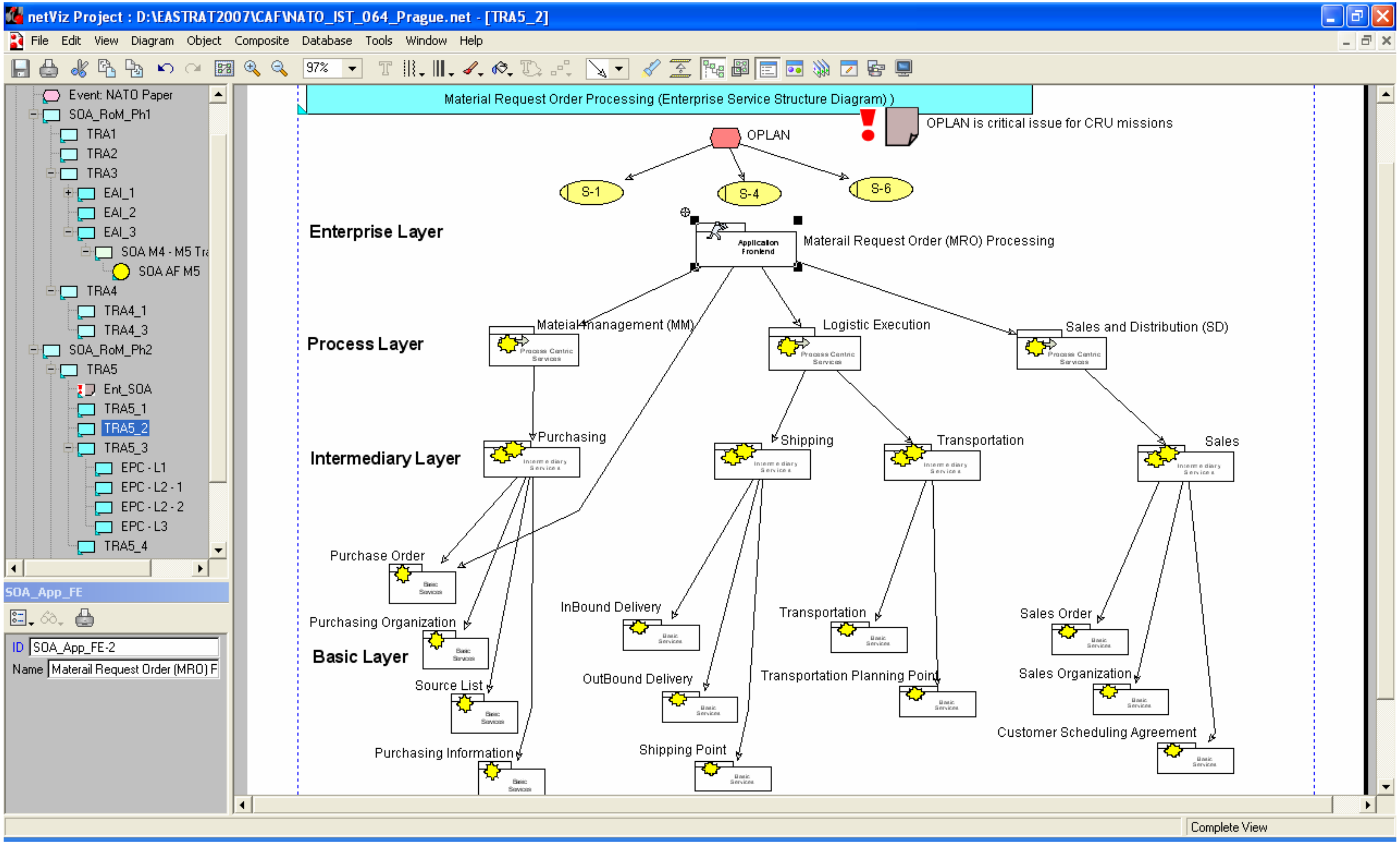
The top layer of SOAs contains application frontends and public enterprise services, which are the end-points that provide access to the SOA. These endpoints facilitate both the communication between end users and the SOA (application frontends) and enable cross-enterprise (for cross-business unit) integration (public enterprise services).

The process layer contains process-centric services - the most advanced service type.

The third layer contains intermediary services. These services act as facades, technology gateways, and adapters. You can also use an intermediary service in order to add functionality to an existing service.

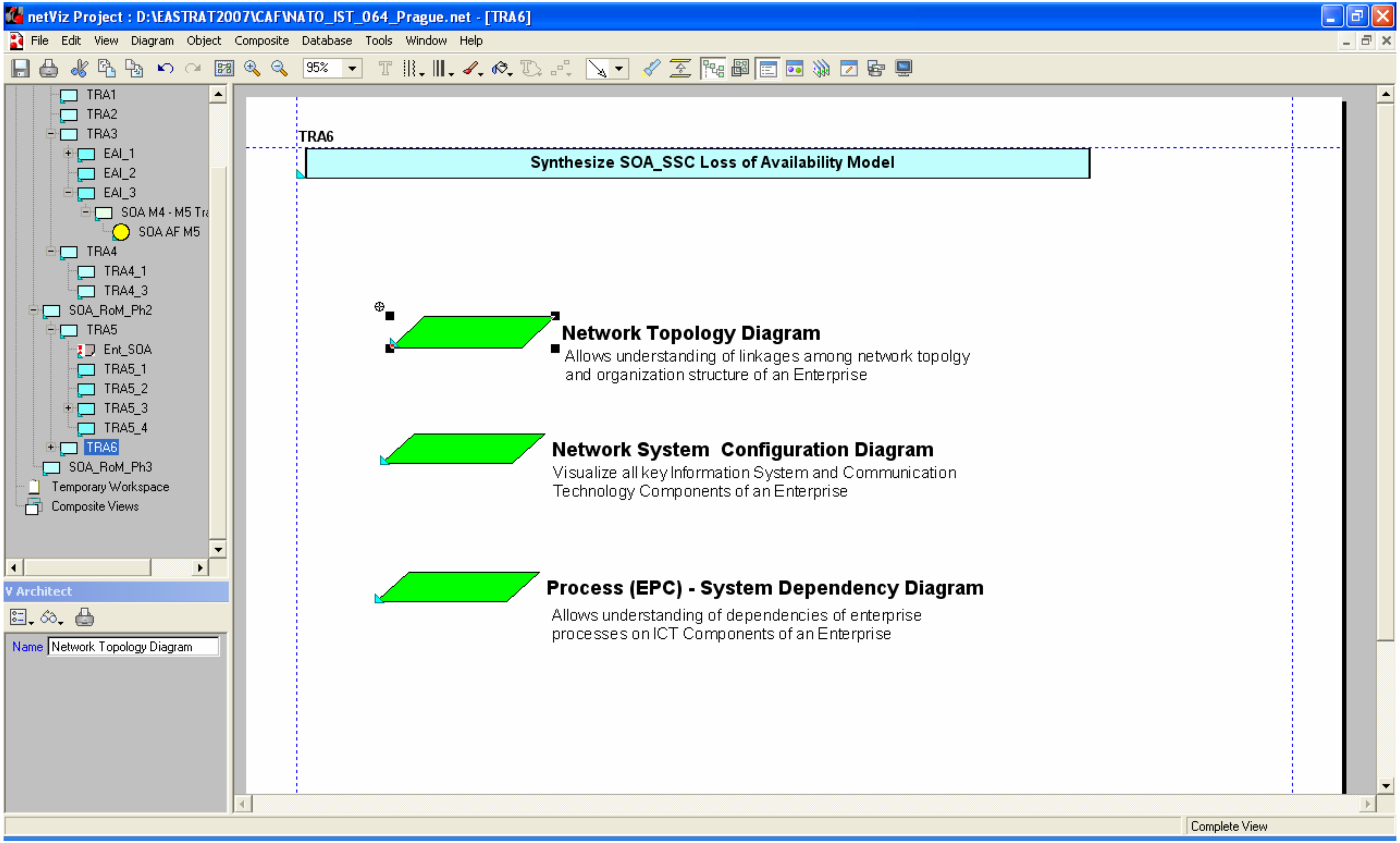
The bottom layer contains the basic services of the SOA. Basic services represent the foundation of the SOA by providing the business logic and data. The basic layer also contains proxies for other companies' public enterprise services.

Figure 12: Enterprise SOA: Services and Layers.



ID SOA_App_FE-2
Name Materail Request Order (MRO) F

Figure 13: Enterprise Service Structure Diagram: MRO Process Decomposition.



Name: Network Topology Diagram

TRA6

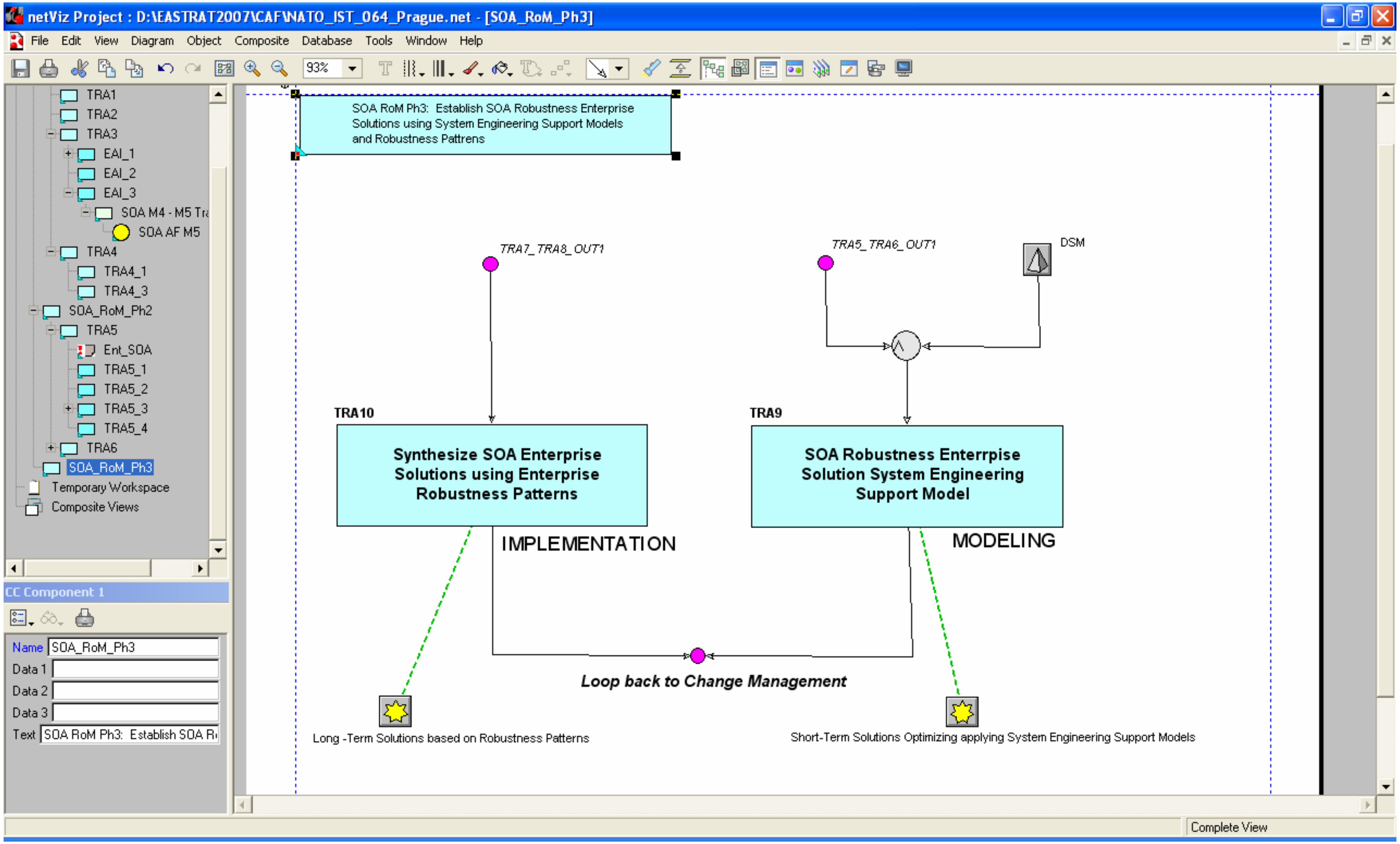
Synthesize SOA_SSC Loss of Availability Model

Network Topology Diagram
Allows understanding of linkages among network topolgy and organization structure of an Enterprise

Network System Configuration Diagram
Visualize all key Information System and Communication Technology Components of an Enterprise

Process (EPC) - System Dependency Diagram
Allows understanding of dependencies of enterprise processes on ICT Components of an Enterprise

Figure 14: Main Parts of the SOA SSC Loss of Availability Model.



netViz Project : D:\EASTRAT2007\CAF\NATO_IST_064_Prague.net - [SOA_RoM_Ph3]

Name: SOA_RoM_Ph3

Text: SOA RoM Ph3: Establish SOA Ri

SOA RoM Ph3: Establish SOA Robustness Enterprise Solutions using System Engineering Support Models and Robustness Pattrens

TRA10

Synthesize SOA Enterprise Solutions using Enterprise Robustness Patterns

TRA9

SOA Robustness Enterrpise Solution System Engineering Support Model

IMPLEMENTATION

MODELING

Loop back to Change Management

Long -Term Solutions based on Robustness Patterns

Short-Term Solutions Optimizing applying System Engineering Support Models

Figure 15: Last Two Tracks of the Robustness Based SOA Roadmap.

High-Availability products are more expensive and require more sophisticated maintenance then standard products. As architects we must be sure that ways how we apply these products improve system stability without negative impacts to enterprise agility.

System engineering let us solving this kind of problems. Loss of Availability Models capture all aspects of process impacts from process owners' perspective and allow architects understanding dependencies of business processes on Enterprise Technology Infrastructure. Risk Analysis and Risk Management (RARM) Models allow us controlling of risk levels on acceptable level. Robustness is one of key issues that participate in risk management. RARM Models in a combination with Decision Support Models (DSM) let us involving Robustness directly into BOCR (Benefits, Opportunities, Cost, and Risk) decision process. All these system engineering models can be further applied during SOA life cycle significantly increasing overall system stability and availability (see Figure 7).

Robustness patterns address primarily bottom two layers in the Enterprise SOA – Basic and Intermediary (Figure 12). These two layers are subject of enterprise application architects' interest [1,10,11]. Patterns of Enterprise Application Architecture can be principally enhanced with robustness methods and properties but it is not easy job. Object oriented programming experience is not enough because requirements for robustness must be derived from dynamic behavior of services in enterprise context. Managing process integrity is a nice example how designers can identify requirements for dynamic behavior of SOA services [17]. Designer starts with analysis of technical failures and business exceptions. Technical Concepts for Robustness can be design using one or more approaches:

- Logging and Tracing.
- ACID (Atomicity, Consistency, Isolation, and Durability) Transactions.
- Transaction Monitors and distributed 2PC (Two-Phase Commit Protocol).
- Nested and Multilevel Transactions.
- Persistent Queues and Transactional Steps.
- Transaction Chains and Compensation.

5.0 CONCLUSION

Robustness based SOA Roadmap is very effective way of management in situations when SOA strategy has harmonizing technological capabilities with business process changes. It allows flexible combination of two kinds of efforts. First, oriented to Short-Time solutions uses power of the System Engineering Support; second approach dives into lower layers of the Enterprise SOA increasing their stability and availability.

The topic discussed in this paper stimulates few interesting research activities for near future:

- Enterprise Integration Model for Scenario Risk Analysis (Figure 3) opens an opportunity to show high available products outcome [20] in the context of enterprise risk analysis and risk management.
- SOA Robustness Decision Support Model (Figure 10) can be developed as etalon model supporting decision makers responsible for SOA deployment.
- SOA Services Scenario Concepts (Figure 11) in a combination with Loss of Availability Model (Figure 14) significantly improve communication between business process owners and robustness designers linking processes understandable to owners with robustness solutions developed by designers.

6.0 REFERENCES

[1] Brown, Whitenack: http://members.aol.com/kgb1001001/Chasms.htm

[2] Burlton, R.T.: "Business Process Management", Sams Publishing, ISBN: 0-672-32063-0.

[3] CCTA: CRAMM Risk Analysis and Management Method, Crown Copyright, CCTA IT Security and Privacy Group, London, 1991, 245.

[4] Cook, S.C.: "The Rise of Systems Engineering within the Australian Defence Organization", IEEE 2004 Proceedings, Singapore, 2004.

[5] El-Gayar, O.F., Fritz, B.D.: "Business Process Re-engineering and Information Security Planning: Opportunities of Integration", SCI 2004 Proceedings, Florida, 2004.

[6] Feglar, T.: "CAF Methodology Usage for Management of Information Systems Protection", SCI 2004, July 18-21, Orlando, Florida (USA).

[7] Feglar, T., Levy, J.: "Protecting Cyber Critical Infrastructures (CCI): Integrating Information Security Risk Analysis and Environmental Vulnerability Analysis", IEEE 2004, October, Singapore.

[8] Feglar, T., Levy, J.: "Dynamic Analytic Network Process: Improving Decision Support Information and Communication Technology", IFORS 2005, July, Honolulu, Hawaii.

[9] Feglar, T.: "ITIL based Service Level Management if SLAs cover Security", CITSA 2004, July, Florida.

[10] Fowler, M.: "Patterns of Enterprise Application Architecture", Addison Wesley, ISBN: 032 11 27420.

[11] Fowler, M.: http://martinfowlercom/ap2/timeNarrative.html

[12] Gelimas, U.J., Sutton, S.J., Fedorowitz, J.: "Business Processes and Information Technology", Thomson South Western, ISBN: 0-324-00878-3.

[13] Hitchins, D.K.: "Advanced Systems Thinking, Engineering, and Management", Artec House, ISBN 1-58053-619-0.

[14] INCOSE: International Council on System Engineering, http://www.incose.org/

[15] ISO / IEC 15288: System Engineering – System Life Cycle Process, ISO, 2002.

[16] Jones, A., Ashenden, D.: "Risk Management for Computer Security", Alsevier, ISBN 0-7506-7795-3.

[17] Krafzig, D., Banke, K., Slama, D.: "Enterprise SOA", Prentice Hall, ISBN 0-13-146575-9.

[18] Michaud, F., Painchaude, F.: "High Availability Solutions to Common Software Failures", NATO Research Workshop IST 064/RW 5011, 2006, November, Prague.

[19] Parasuraman, Zeithaml, A., Berry, L.L.: "A Conceptual Model of Service Quality and its Implication for Future Research", Journal of Marketing, Vol. 49, 1985, pp. 41-50.

[20] Saaty, T.L.: "The Analytic Network Process – Decision Making With Dependence and Feedback", RWS Publications, Pittsburgh, ISBN 0-9620317-9-8.

[21] Scheer, A.V.: “ARIS for SAP NetWeaver: The Business Process Design Solution for SAP NetWeaver”, IDS Scheer, 2005.

[22] Zeithaml, V.A., Bitner, M.J.: “Service Marketing”, McGraw-Hill, New York, NY, 1996, p. 700.

Appendix 1: The Decision Support Model for a Choice of "The Best Solution Based on SOA Robustness"

Appendix 1 describes very simple AHP model that allows choosing the most optimal alternative using criteria for their comparison.

Model structure is depicted in the Figure 1. Alternatives represent high-availability products described by Frederic Michaud and Frederic Painchaud ("High-Availability Solutions to Common Software Failures"). In accordance with AHP methodology we firstly design preferences for chosen criteria and then we compare alternatives across each criteria respectively sub-criteria.

Figures 2 – 7 show characteristic snapshots used by decision makers.

Figure 2 shows finial model in which all preferences were successfully calculated and checked for integrity.

Figure 3 visualize final preferences; the most preferable high-availability solution is alternative A2 – Microsoft Clustering Services; the second best is alternative A1 – Marathon everRun.

Figure 4 illustrates very different performance of alternatives for different criteria. For example alternative A2 has very high performance in criteria C2 (Adding Value), but very low in criteria C4 (Cost). Alternative A5 (Linux) has very low performance in criteria C1 but very high in criteria C4.

In situation like just described decision makers want to avoid mistakes that relate to criteria preferences calculations. Dynamic graphs allow us elaboration with other criteria preferences.

Figure 5 depicts starting situation (Alternative A2 is the most preferable).

Figure 6 illustrates the situation when decision maker increases preference for criteria C1 (Availability Improvement) from 19.1 % to 28.5 %. We can see that preferences of alternatives become different – the winner is alternative A1.

In figure 7 we increased preference for criteria C5 (Cost) from 10.9 % to 33.2 %. A1 is still the winner but followed by alternative A5 (Linux).

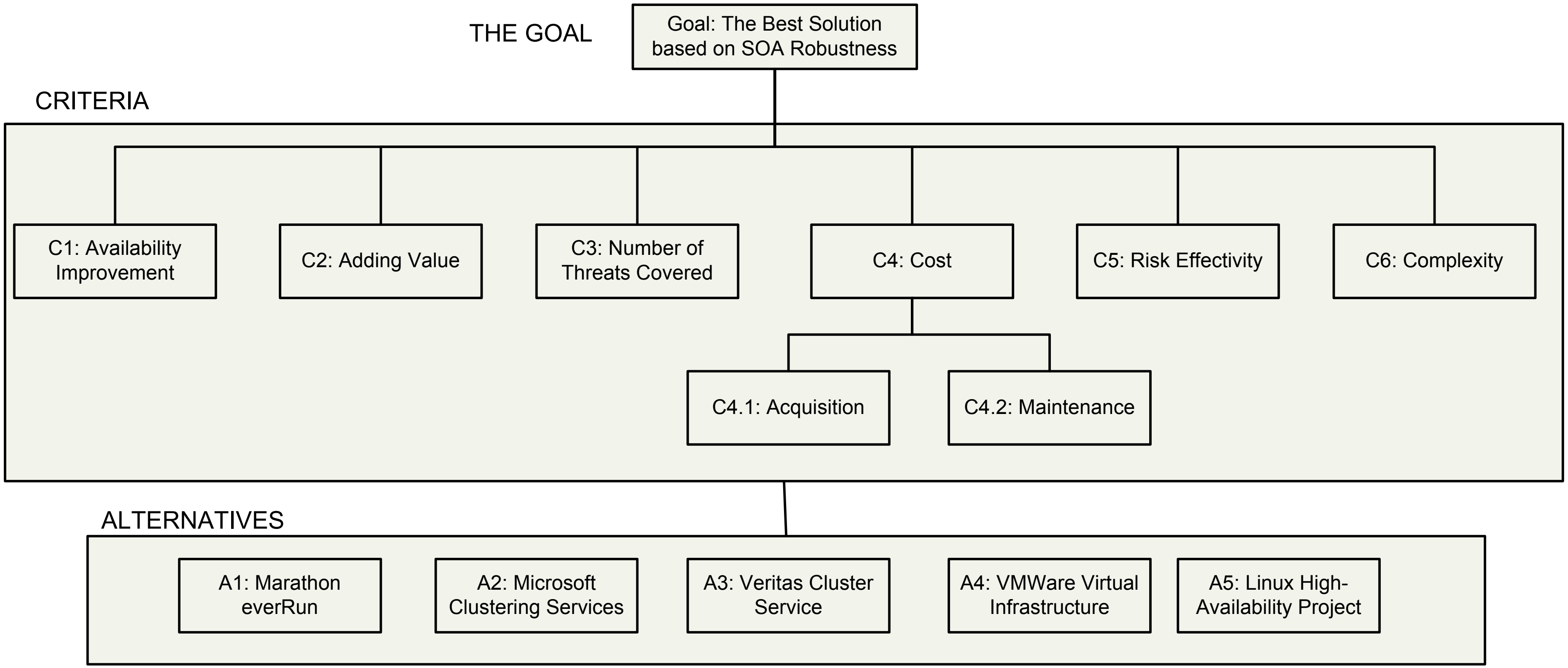
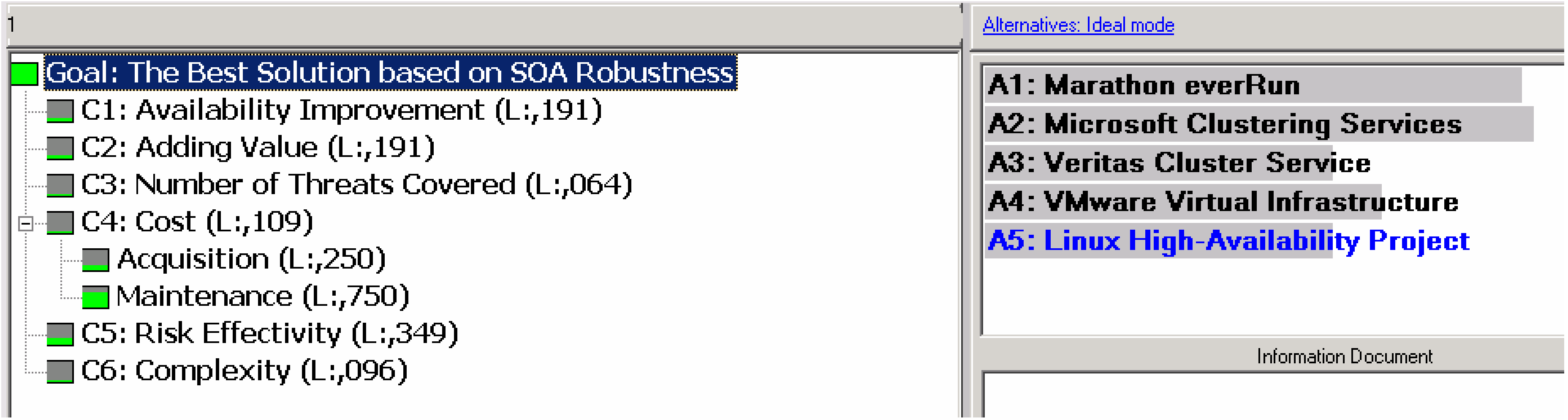

THE GOAL
Goal: The Best Solution based on SOA Robustness
CRITERIA
C1: Availability Improvement
C2: Adding Value
C3: Number of Threats Covered
C4: Cost
C5: Risk Effectivity
C6: Complexity
C4.1: Acquisition
C4.2: Maintenance
ALTERNATIVES
A1: Marathon everRun
A2: Microsoft Clustering Services
A3: Veritas Cluster Service
A4: VMWare Virtual Infrastructure
A5: Linux High-Availability Project

Figure 1: Decision Model Structure.



1

Goal: The Best Solution based on SOA Robustness
- C1: Availability Improvement (L:,191)
- C2: Adding Value (L:,191)
- C3: Number of Threats Covered (L:,064)
- C4: Cost (L:,109)
  - Acquisition (L:,250)
  - Maintenance (L:,750)
- C5: Risk Effectivity (L:,349)
- C6: Complexity (L:,096)

Alternatives: Ideal mode

A1: Marathon everRun

A2: Microsoft Clustering Services

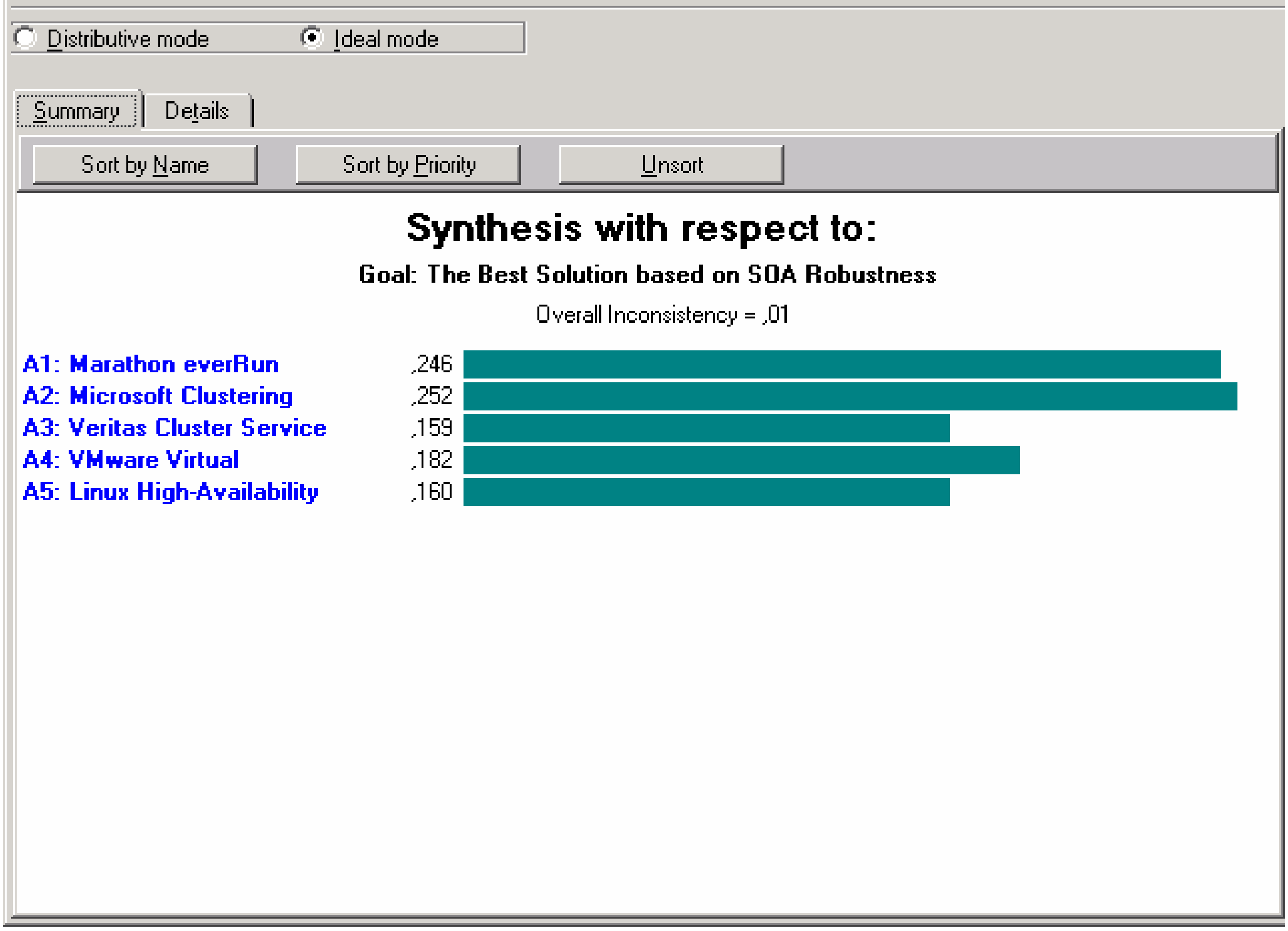
A3: Veritas Cluster Service

A4: VMware Virtual Infrastructure

A5: Linux High-Availability Project

Information Document

Figure 2: Final Decision Support Model in which All Calculations were Successfully Finished.



○ Distributive mode ● Ideal mode

Synthesis with respect to:

Goal: The Best Solution based on SOA Robustness

Overall Inconsistency = ,01

A1: Marathon everRun	,246
A2: Microsoft Clustering	,252
A3: Veritas Cluster Service	,159
A4: VMware Virtual	,182
A5: Linux High-Availability	,160

Figure 3: The Winner Alternative is A1 – Microsoft Clustering Services; the Second Best Alternative is Marathon everRun. We want to know more about decision making process and we use Performance graph in accordance with next figure.

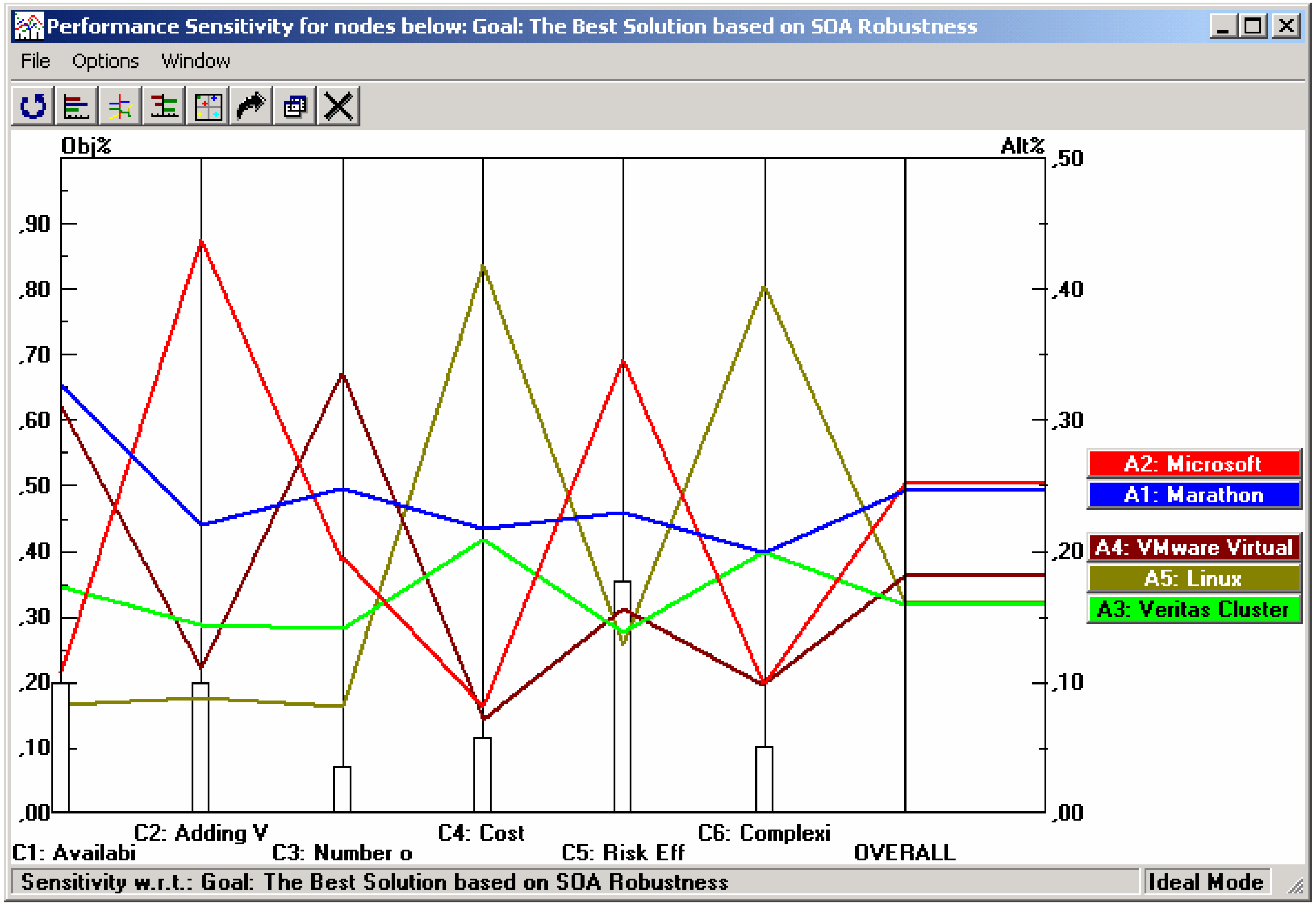


Figure 4: Performance Graph Shows very Different Behavior of Alternatives in Dependency on Particular Criteria. Because alternatives are so heavily dependent on criteria and their preferences we want to know how situation could change in the case that we change preferences among criteria.

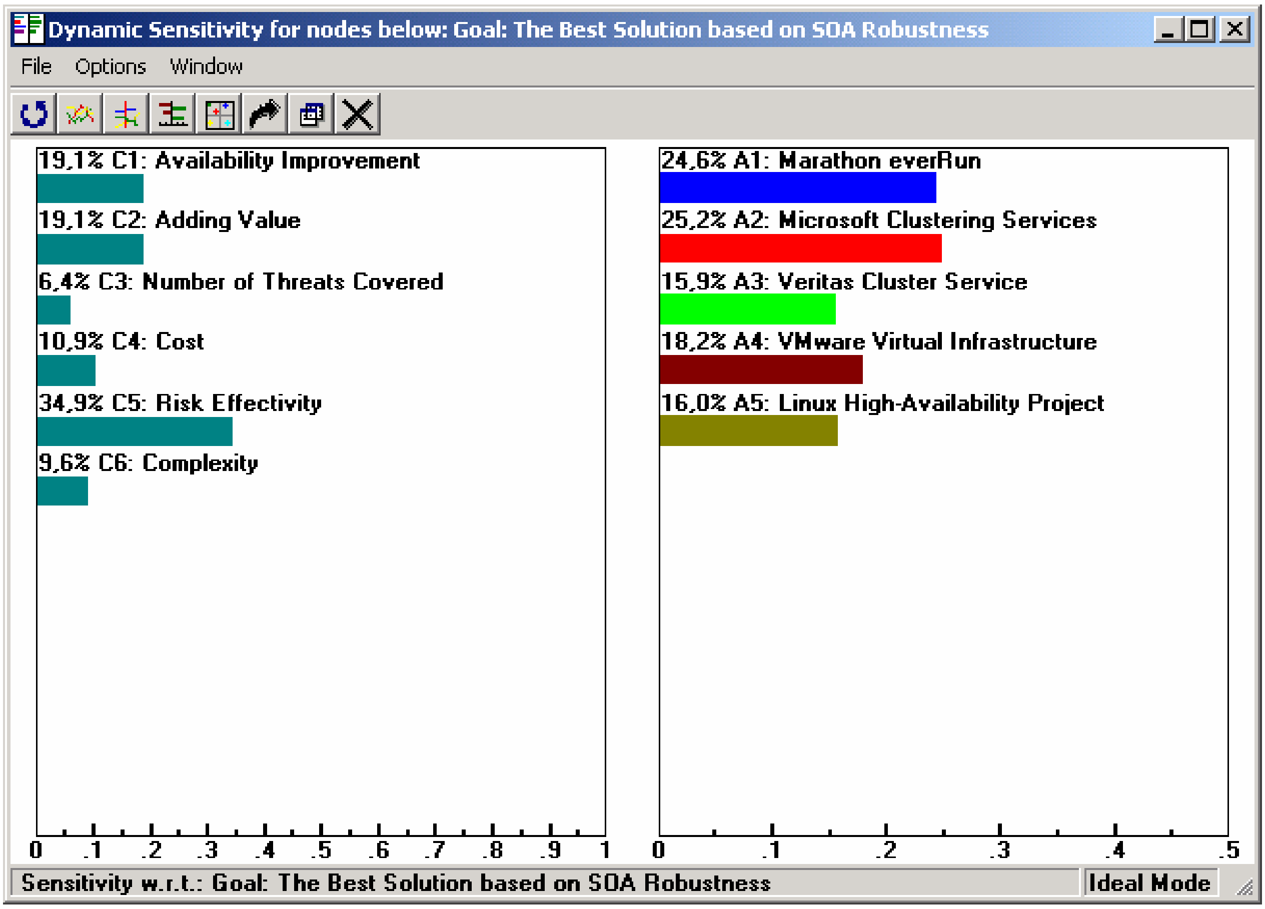
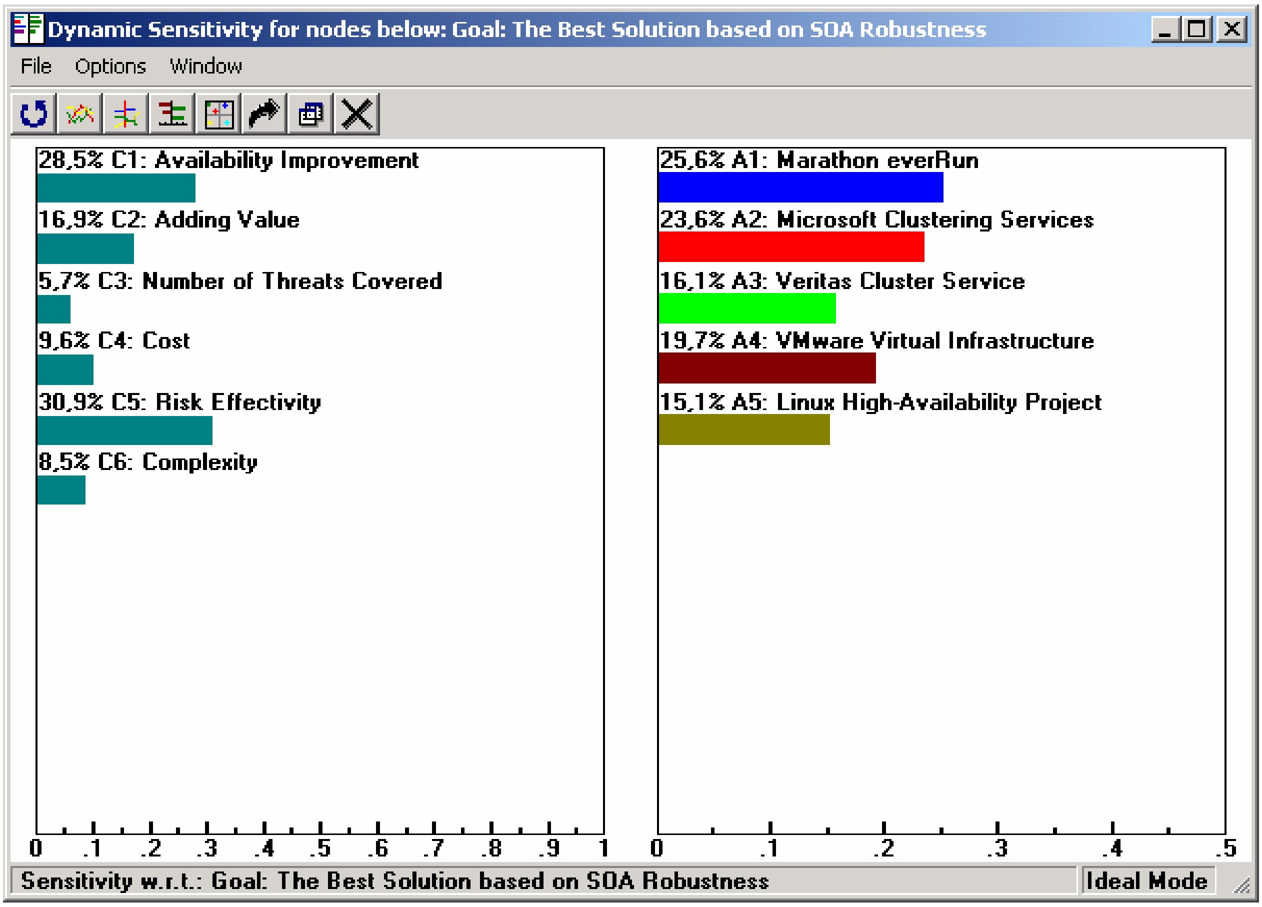

Dynamic Sensitivity for nodes below: Goal: The Best Solution based on SOA Robustness
File Options Window
19,1% C1: Availability Improvement
19,1% C2: Adding Value
6,4% C3: Number of Threats Covered
10,9% C4: Cost
34,9% C5: Risk Effectivity
9,6% C6: Complexity
0 .1 .2 .3 .4 .5 .6 .7 .8 .9 1
24,6% A1: Marathon everRun
25,2% A2: Microsoft Clustering Services
15,9% A3: Veritas Cluster Service
18,2% A4: VMware Virtual Infrastructure
16,0% A5: Linux High-Availability Project
0 .1 .2 .3 .4 .5
Sensitivity w.r.t.: Goal: The Best Solution based on SOA Robustness
Ideal Mode

Figure 5: Dynamic Graph let us Observe Influence how Criteria Preferences could Change Results of Our Decision. In this case we consider criteria preferences that results situation described in the Figure 3 (A2 winner, A1 second best).



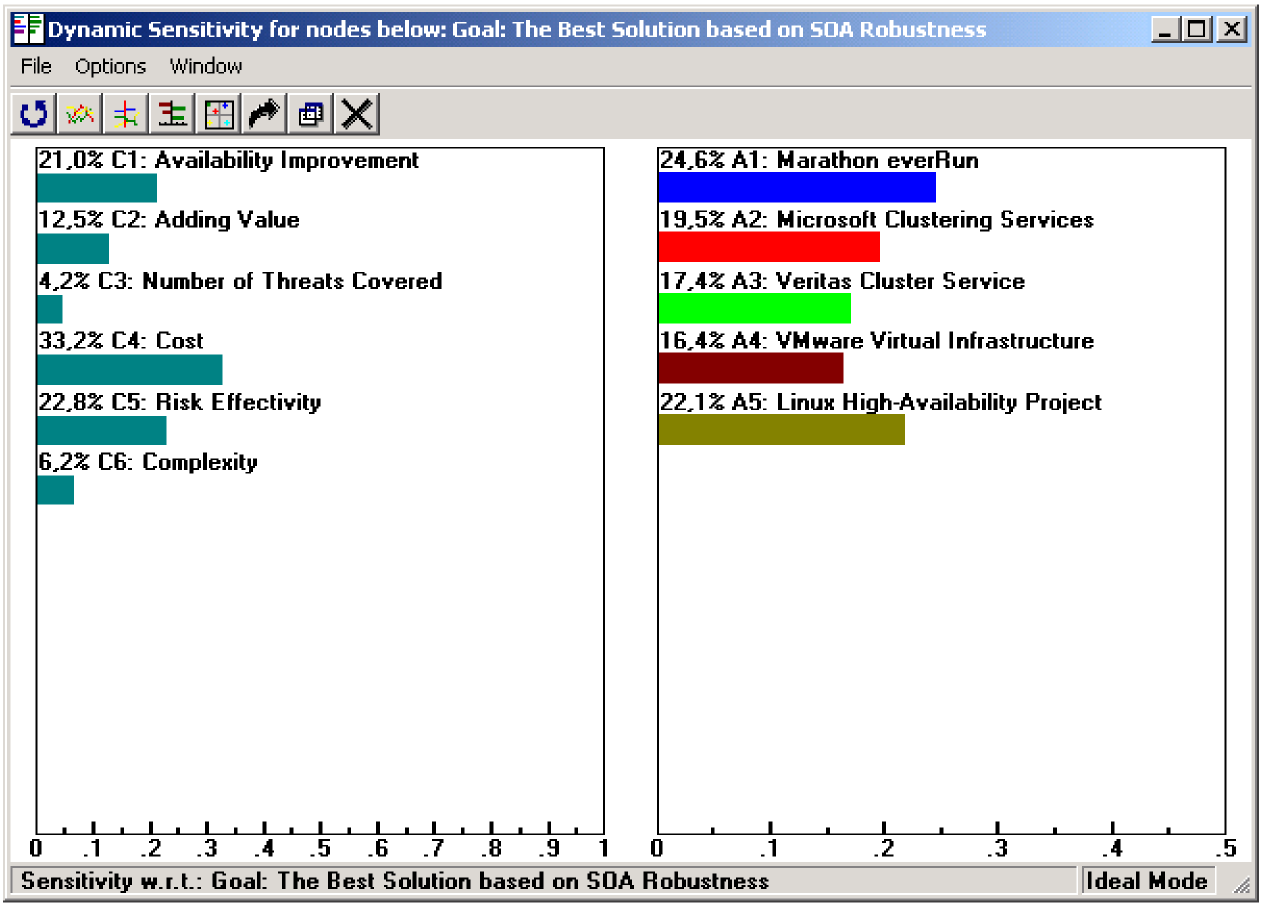
Dynamic Sensitivity for nodes below: Goal: The Best Solution based on SOA Robustness

File
Options
Window
28,5% C1: Availability Improvement
16,9% C2: Adding Value
5,7% C3: Number of Threats Covered
9,6% C4: Cost
30,9% C5: Risk Effectivity
8,5% C6: Complexity
0
.1
.2
.3
.4
.5
.6
.7
.8
.9
1
25,6% A1: Marathon everRun
23,6% A2: Microsoft Clustering Services
16,1% A3: Veritas Cluster Service
19,7% A4: VMware Virtual Infrastructure
15,1% A5: Linux High-Availability Project
0
.1
.2
.3
.4
.5

Sensitivity w.r.t.: Goal: The Best Solution based on SOA Robustness

Ideal Mode

Figure 6: We have Increased Preference of the Criteria C1 (Availability Improvement) from 19.1 % to 28,5 %. We can see that preferences of alternatives become different – the winner is alternative A1.



Dynamic Sensitivity for nodes below: Goal: The Best Solution based on SOA Robustness
File
Options
Window
0
.1
.2
.3
.4
.5
.6
.7
.8
.9
1
0
.1
.2
.3
.4
.5
Ideal Mode

21,0% C1: Availability Improvement

12,5% C2: Adding Value

4,2% C3: Number of Threats Covered

33,2% C4: Cost

22,8% C5: Risk Effectivity

6,2% C6: Complexity

24,6% A1: Marathon everRun

19,5% A2: Microsoft Clustering Services

17,4% A3: Veritas Cluster Service

16,4% A4: VMware Virtual Infrastructure

22,1% A5: Linux High-Availability Project

Sensitivity w.r.t.: Goal: The Best Solution based on SOA Robustness

Figure 7: We have Increased Preference of the Criteria C5 (Cost) from 10.9 % to 33.2 %. A1 is still the winner but followed by alternative A5 (Linux).

Appendix 2: The Material Request Order (MRO) Process Sample

Appendix 2 describes scenario for MRO processing and how this scenario is applied for a synthesize of the Loss of Availability model. This sample let us understanding of tracks TRA5 and TRA 6 in the Robustness based SOA Roadmap.

Each scenario is developed in two synchronized tracks – TRA5 and TRA6 (Figure 1).

Figures 2 – 8 show snapshots characterizing SOA Services Scenario Concept (SOA SSC). Figures 9 – 13 show snapshots characterizing SOA SSC Loss of Availability model.

Figure 2. Four main parts of the SOA SCC – Organization Diagram, Enterprise Service Structure Diagram, EPC Diagram, and Service Interaction Diagram.

Figure 3. SOA SCC Organizing Diagram captures all key actors (organization units) involved in the MRO processing.

Figure 4. Enterprise Service Structure Diagram let us understanding SOA services hierarchy that must be available for the MRO processing.

Figure 5. EPC Diagram visualizes information and material flows among various organizational levels involved in the MRO processing. EPC diagrams are also applied for modeling inside each organizational level (see next figures).

Figure 6. Request for Material Order (MRO) appears at the tactical level and it is created by logisticians in the CRU Battalion.

Figure 7. The first superordinate organization unit that reacts to the battalion's MRO is the CRU Brigade.

Figure 8. Activities described as green blocks in EPC diagrams are supported by SOA services functionality that is a result of interaction among particular SOA services in accordance with Service Interaction Diagrams.

Last five snapshots deal with the SOA SSC Loss of Availability model.

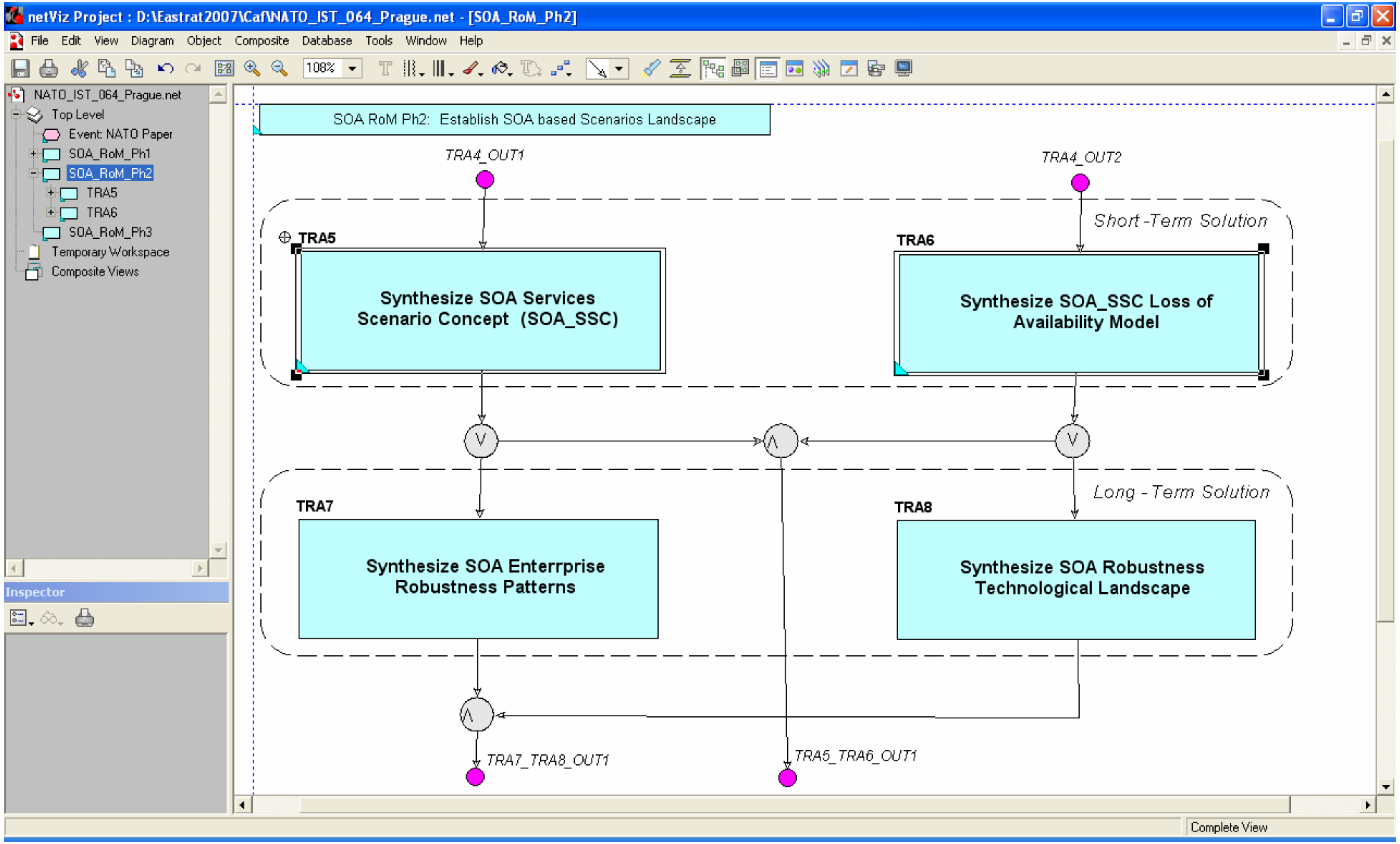
Figure 9. Three main parts of the SOA SSC Loss of Availability model – Network System Configuration Diagram, Network Topology Diagram, and Process (EPC) – System Dependency Diagram.

Figure 10. Network System Configuration Diagram captures all ICT components that must be available for the MRO processing. ICT components are associated with network nodes.

Figure 11. Network Topology Diagram captures all organization units involved in the MRO processing. Organization units are associated with network nodes.

Figure 12. Process (EPC) – System Dependency Diagram copies EPC Diagram used for visualization of the MRO processing (see Figure 5).

Figure 13. Process (EPC) – System Dependency Diagram is applied for each organizational level involved in the MRO processing (Figure 11). It allows understanding of dependencies among events, functions, and information objects (services) and ICT components.

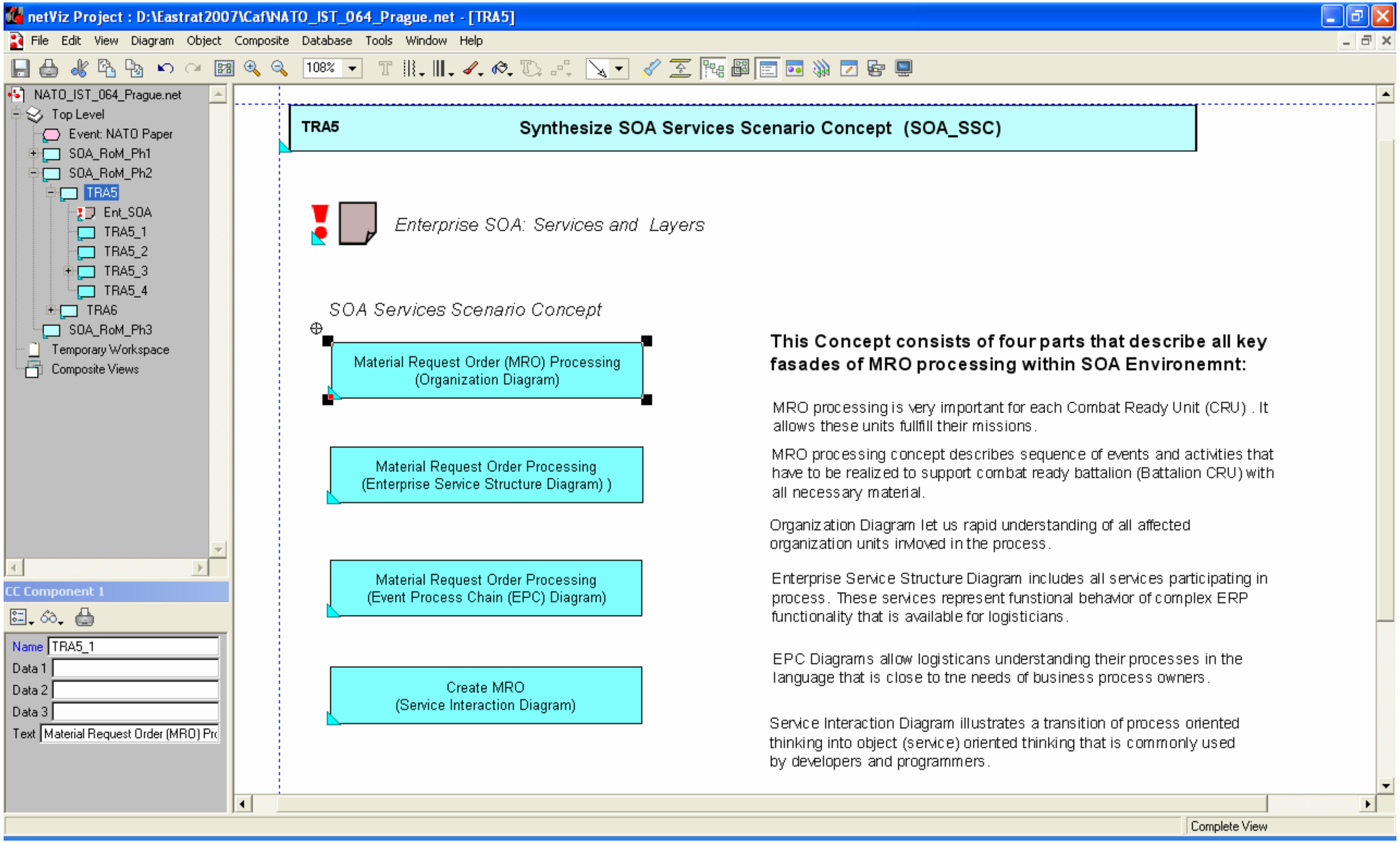


netViz Project : D:\Eastrat2007\Caf\NATO_IST_064_Prague.net - [SOA_RoM_Ph2]
File Edit View Diagram Object Composite Database Tools Window Help
108%
NATO_IST_064_Prague.net
Top Level
Event: NATO Paper
SOA_RoM_Ph1
SOA_RoM_Ph2
TRA5
TRA6
SOA_RoM_Ph3
Temporary Workspace
Composite Views
Inspector
Complete View

SOA RoM Ph2: Establish SOA based Scenarios Landscape

TRA4_OUT1
TRA4_OUT2
Short -Term Solution
TRA5
Synthesize SOA Services Scenario Concept (SOA_SSC)
TRA6
Synthesize SOA_SSC Loss of Availability Model
V
Λ
V
Long - Term Solution
TRA7
Synthesize SOA Enterrprise Robustness Patterns
TRA8
Synthesize SOA Robustness Technological Landscape
Λ
TRA7_TRA8_OUT1
TRA5_TRA6_OUT1

Figure 1: Each SOA Services Scenario is Developed within Two Synchronized Tracks TRA5 and TRA6.



Name TRA5_1
Data 1
Data 2
Data 3
Text Material Request Order (MRO) Prc

TRA5 Synthesize SOA Services Scenario Concept (SOA_SSC)

Enterprise SOA: Services and Layers

SOA Services Scenario Concept

Material Request Order (MRO) Processing
(Organization Diagram)

Material Request Order Processing
(Enterprise Service Structure Diagram))

Material Request Order Processing
(Event Process Chain (EPC) Diagram)

Create MRO
(Service Interaction Diagram)

This Concept consists of four parts that describe all key fasades of MRO processing within SOA Environemnt:

MRO processing is very important for each Combat Ready Unit (CRU) . It allows these units fullfill their missions.

MRO processing concept describes sequence of events and activities that have to be realized to support combat ready battalion (Battalion CRU) with all necessary material.

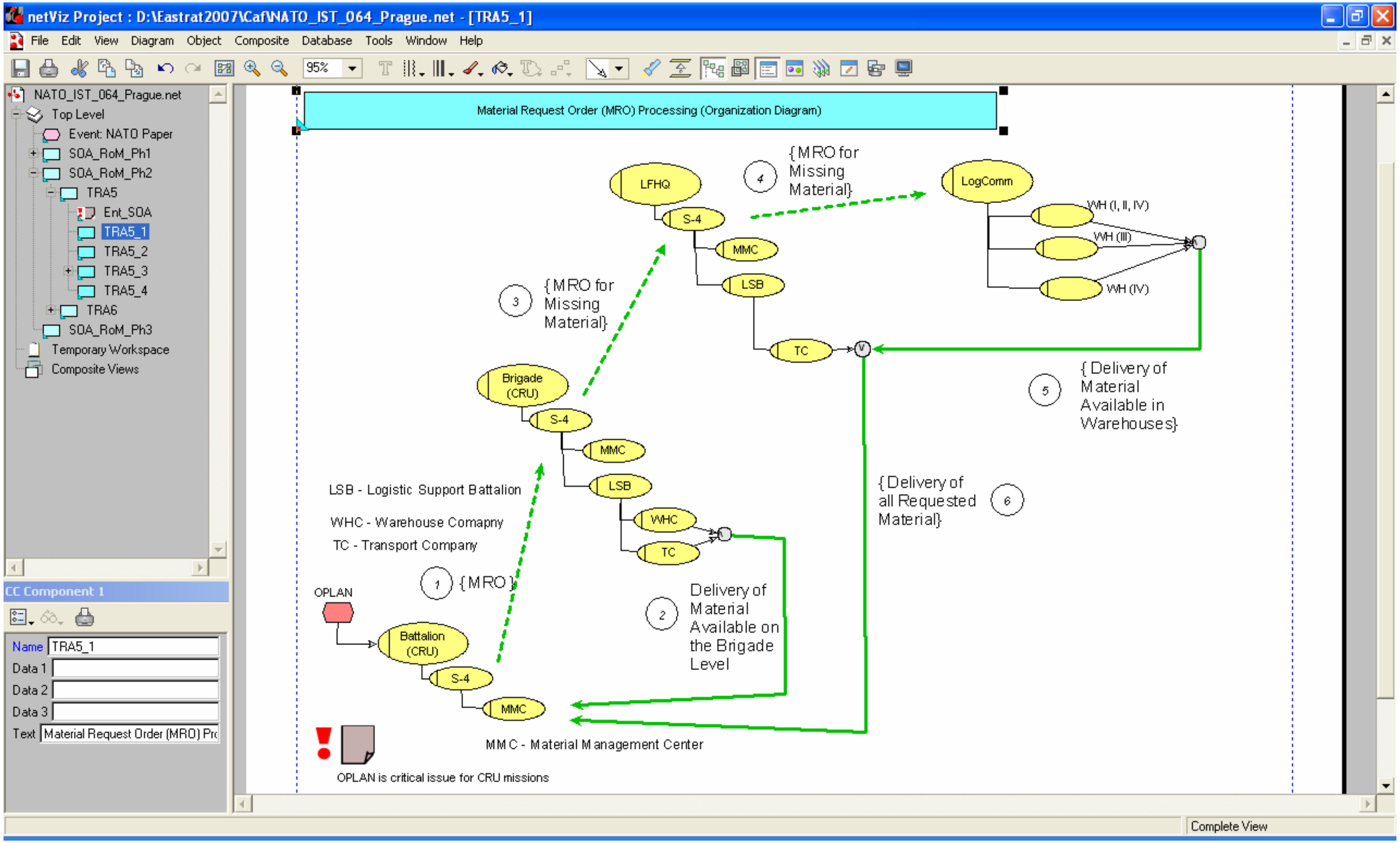
Organization Diagram let us rapid understanding of all affected organization units involved in the process.

Enterprise Service Structure Diagram includes all services participating in process. These services represent funstional behavior of complex ERP functionality that is available for logisticians.

EPC Diagrams allow logisticans understanding their processes in the language that is close to tne needs of business process owners.

Service Interaction Diagram illustrates a transition of process oriented thinking into object (service) oriented thinking that is commonly used by developers and programmers.

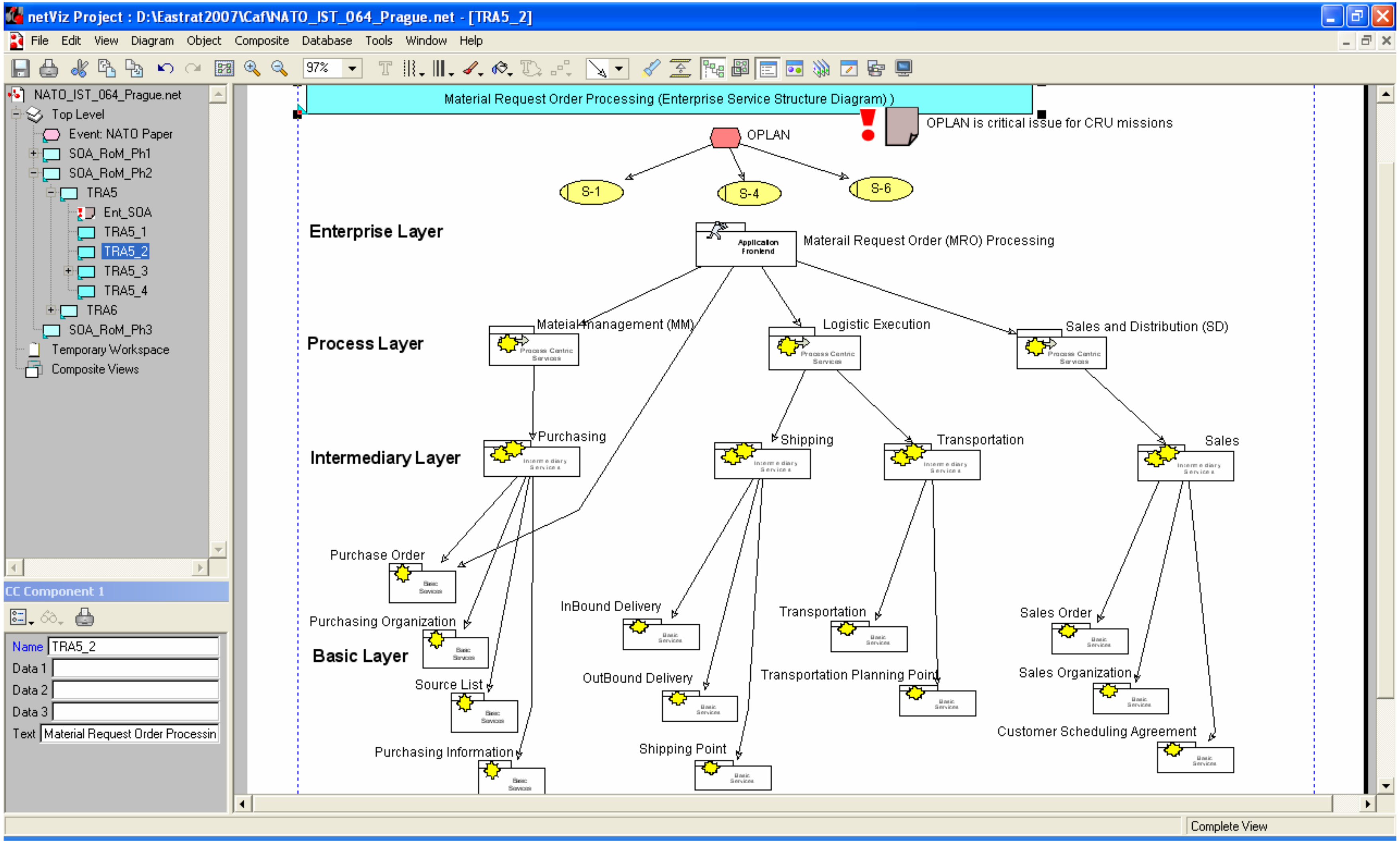
Figure 2: Four Main Parts of the SOA SCC – Organization Diagram, Enterprise Service Structure Diagram, EPC Diagram, and Service Interaction Diagram.



netViz Project : D:\Eastrat2007\Caf\NATO_IST_064_Prague.net - [TRA5_1]

Name: TRA5_1
Data 1:
Data 2:
Data 3:
Text: Material Request Order (MRO) Pr

Figure 3: SOA SCC Organizing Diagram Captures All Key Actors (Organization Units) Involved in the MRO Processing.



Name TRA5_2

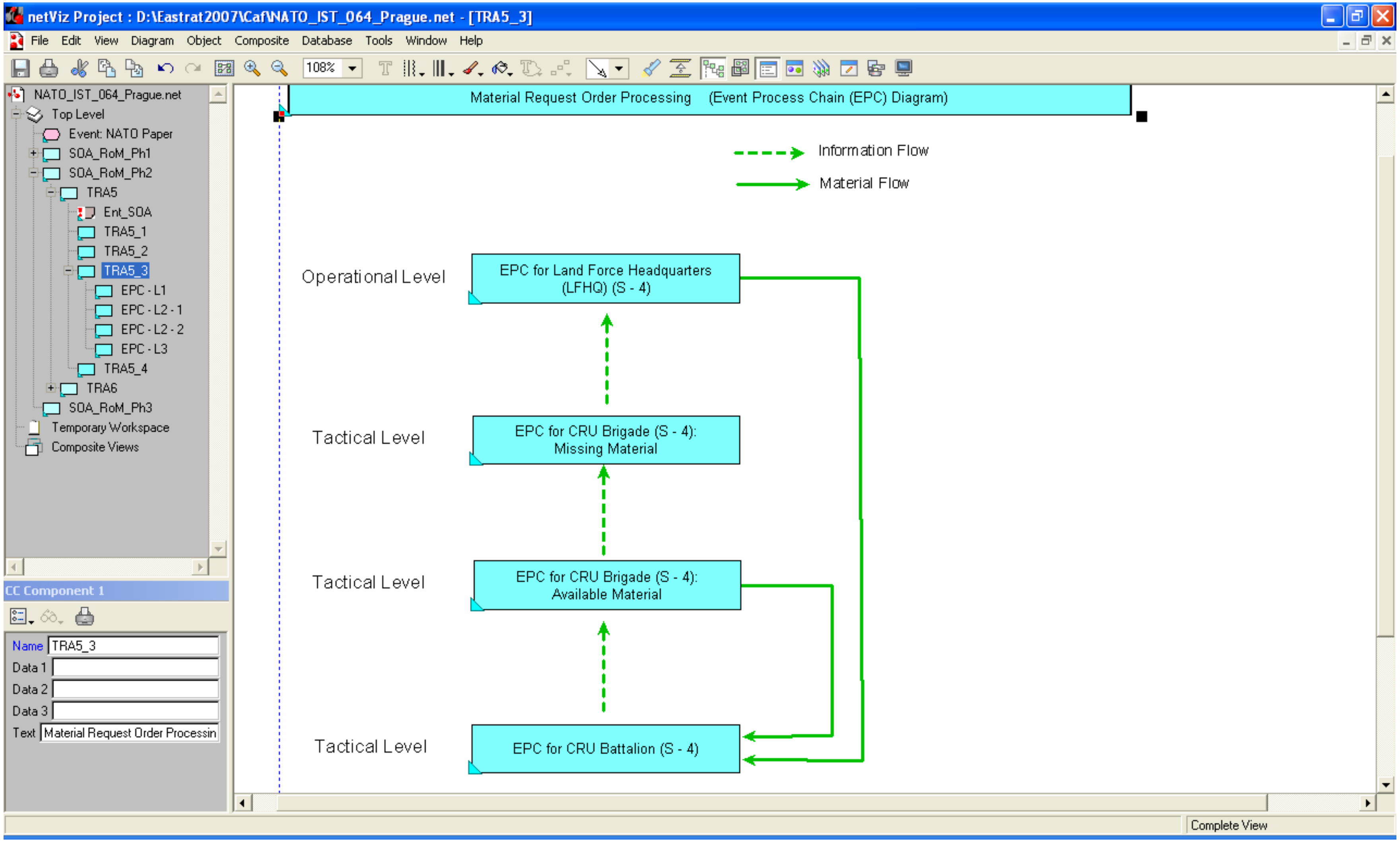
Data 1

Data 2

Data 3

Text Material Request Order Processin

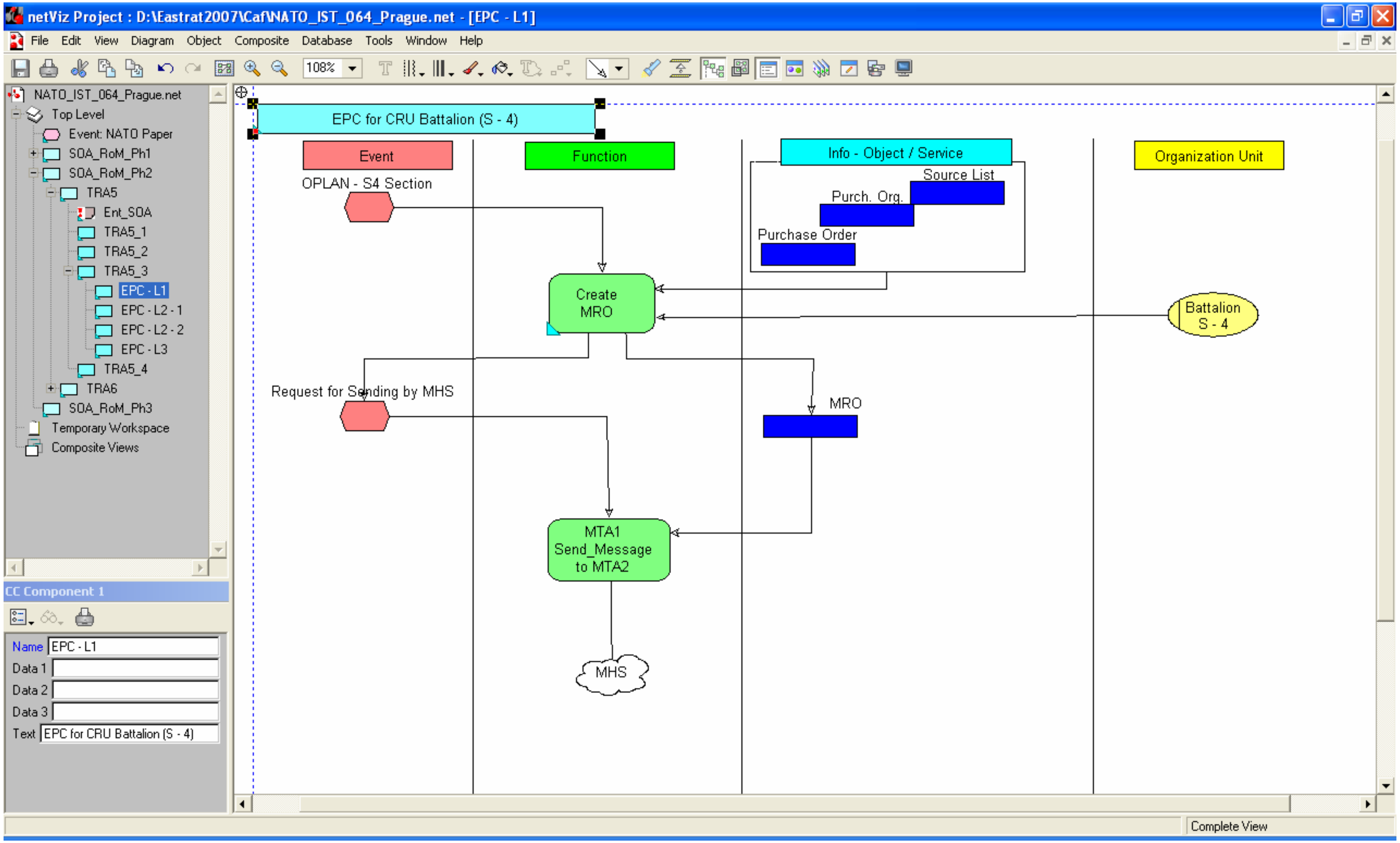
Figure 4: Enterprise Service Structure Diagram let us Understanding SOA Services Hierarchy that Must be Available for the MRO Processing.


netViz Project : D:\Eastrat2007\Caf\NATO_IST_064_Prague.net - [TRA5_3]
File Edit View Diagram Object Composite Database Tools Window Help
108%
NATO_IST_064_Prague.net
Top Level
Event: NATO Paper
SOA_RoM_Ph1
SOA_RoM_Ph2
TRA5
Ent_SOA
TRA5_1
TRA5_2
TRA5_3
EPC - L1
EPC - L2 - 1
EPC - L2 - 2
EPC - L3
TRA5_4
TRA6
SOA_RoM_Ph3
Temporary Workspace
Composite Views
CC Component 1
Complete View

Name TRA5_3
Data 1
Data 2
Data 3
Text Material Request Order Processin

Material Request Order Processing (Event Process Chain (EPC) Diagram)
Information Flow
Material Flow
Operational Level
EPC for Land Force Headquarters (LFHQ) (S - 4)
Tactical Level
EPC for CRU Brigade (S - 4): Missing Material
Tactical Level
EPC for CRU Brigade (S - 4): Available Material
Tactical Level
EPC for CRU Battalion (S - 4)

Figure 5: EPC Diagram Visualizes Information and Material Flows among Various Organizational Levels Involved in the MRO Processing. EPC diagrams are also applied for modeling inside each organizational level (see next figures).



Name EPC - L1

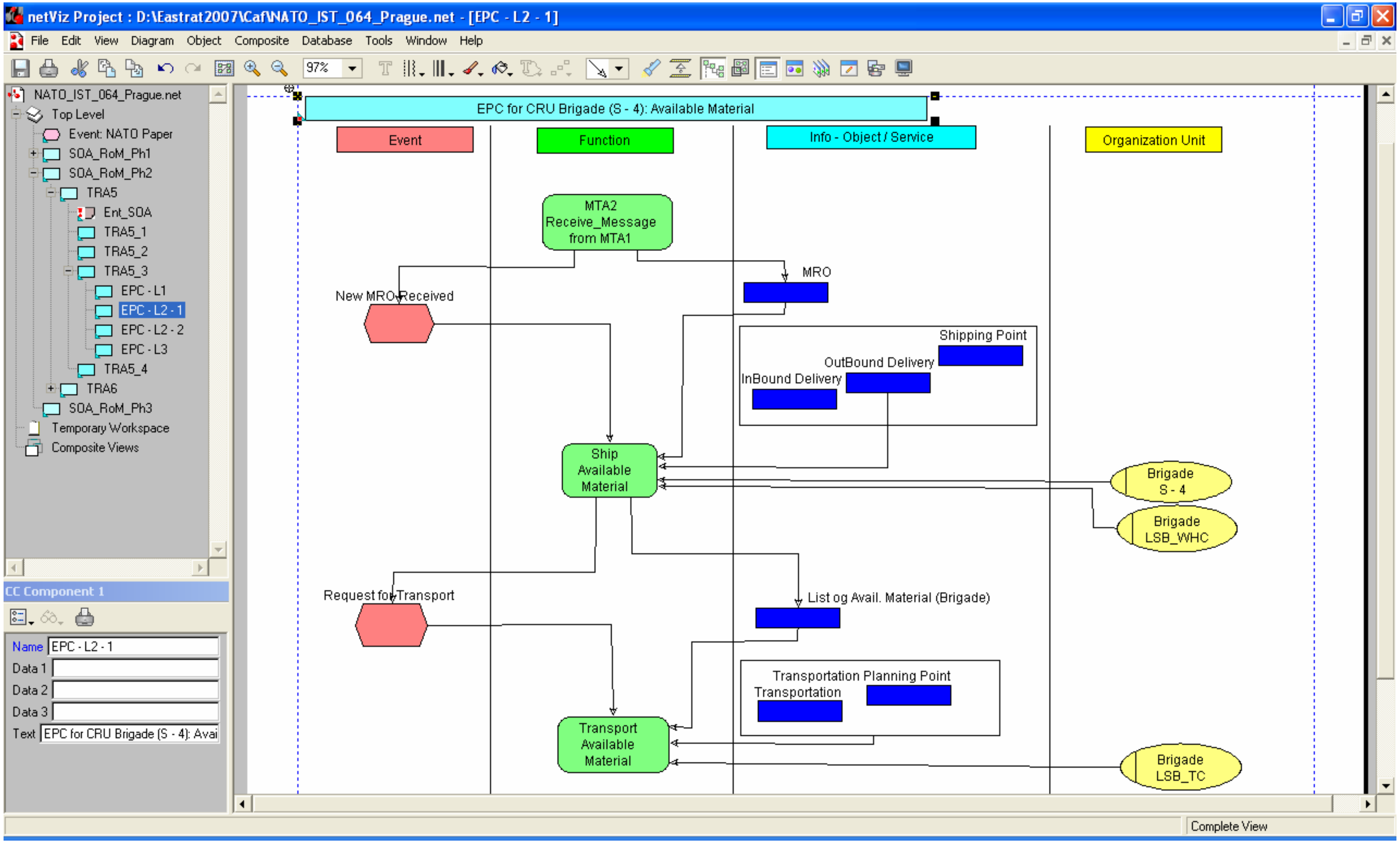
Data 1

Data 2

Data 3

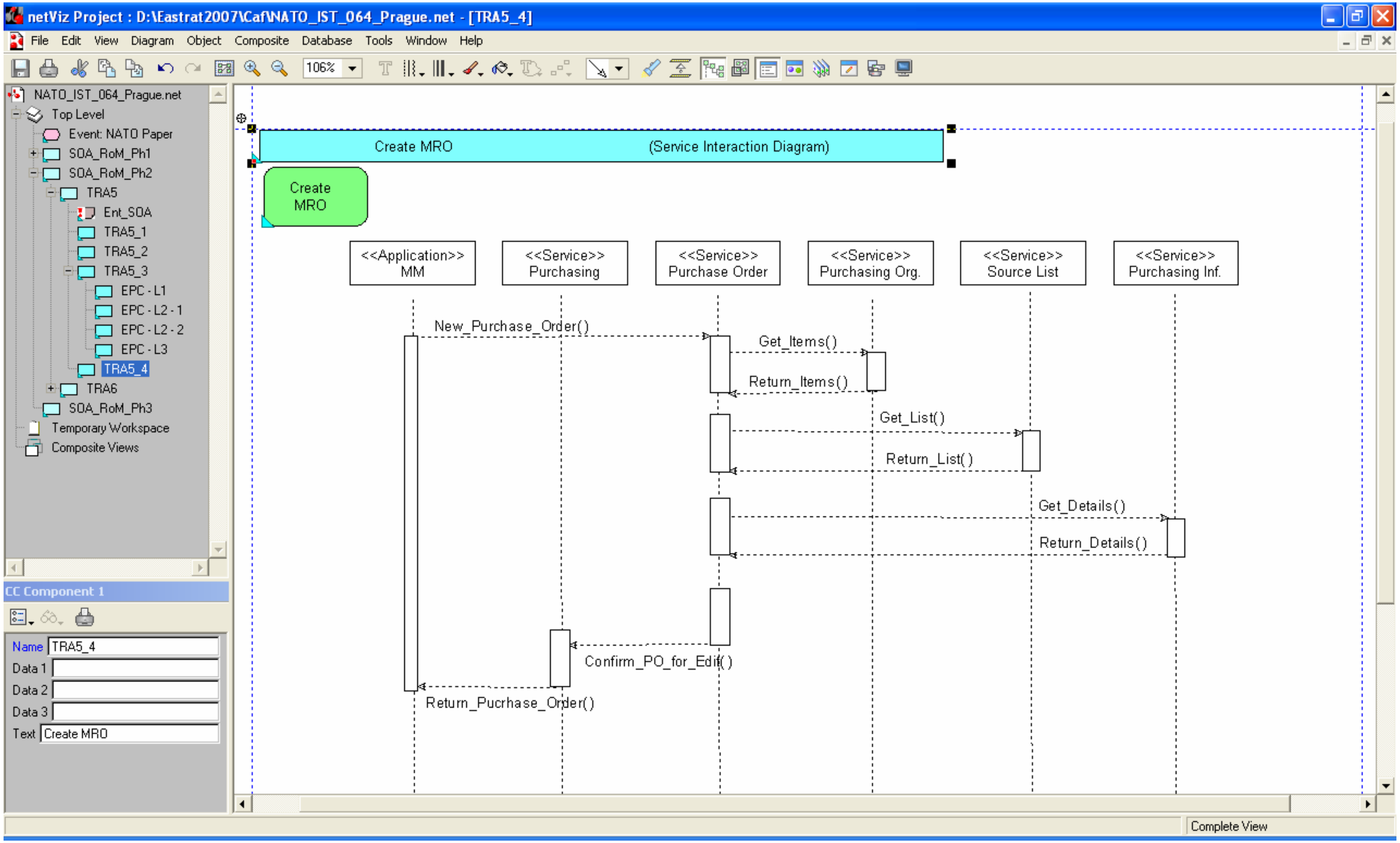
Text EPC for CRU Battalion (S - 4)

Figure 6: Request for Material Order (MRO) Appears at the Tactical Level and it is Created by Logisticians in the CRU Battalion.



Name EPC - L2 - 1
Data 1
Data 2
Data 3
Text EPC for CRU Brigade (S - 4): Avai

Figure 7: The First Superordinate Organization Unit that Reacts to the Battalion’s MRO is the CRU Brigade.



netViz Project : D:\Eastrat2007\Caf\NATO_IST_064_Prague.net - [TRA5_4]

Name TRA5_4

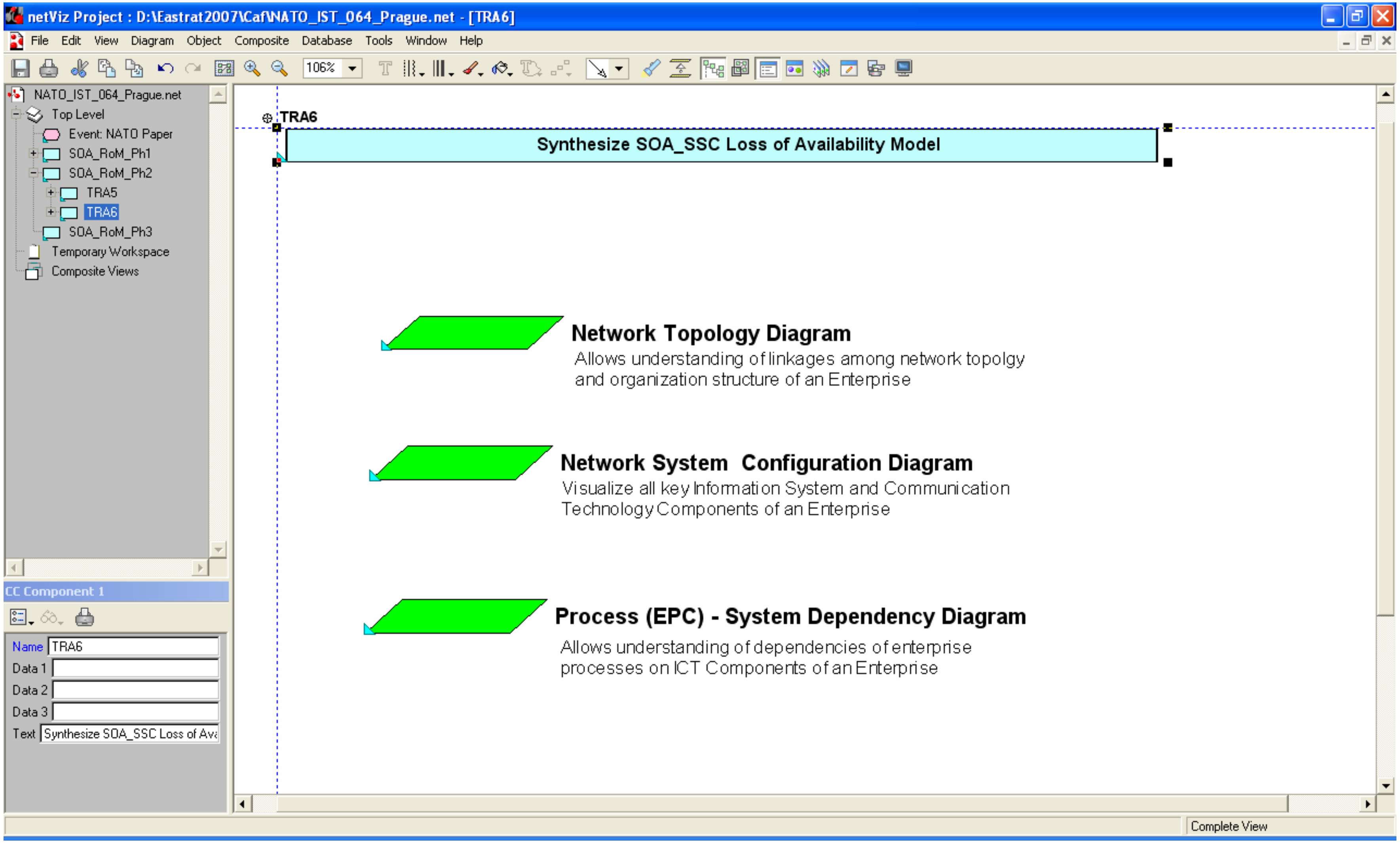
Data 1

Data 2

Data 3

Text Create MRO

Figure 8: Activities Described as Green Blocks in EPC Diagrams are Supported by SOA Services Functionality that is a Result of Interaction among Particular SOA Services in Accordance with Service Interaction Diagrams.



Name: TRA6
Data 1:
Data 2:
Data 3:
Text: Synthesize SOA_SSC Loss of Ava

Synthesize SOA_SSC Loss of Availability Model

Network Topology Diagram
Allows understanding of linkages among network topolgy and organization structure of an Enterprise

Network System Configuration Diagram
Visualize all key Information System and Communication Technology Components of an Enterprise

Process (EPC) - System Dependency Diagram
Allows understanding of dependencies of enterprise processes on ICT Components of an Enterprise

Figure 9: Three Main Parts of the SOA SSC Loss of Availability Model – Network System Configuration Diagram, Network Topology Diagram, and Process (EPC) – System Dependency Diagram.

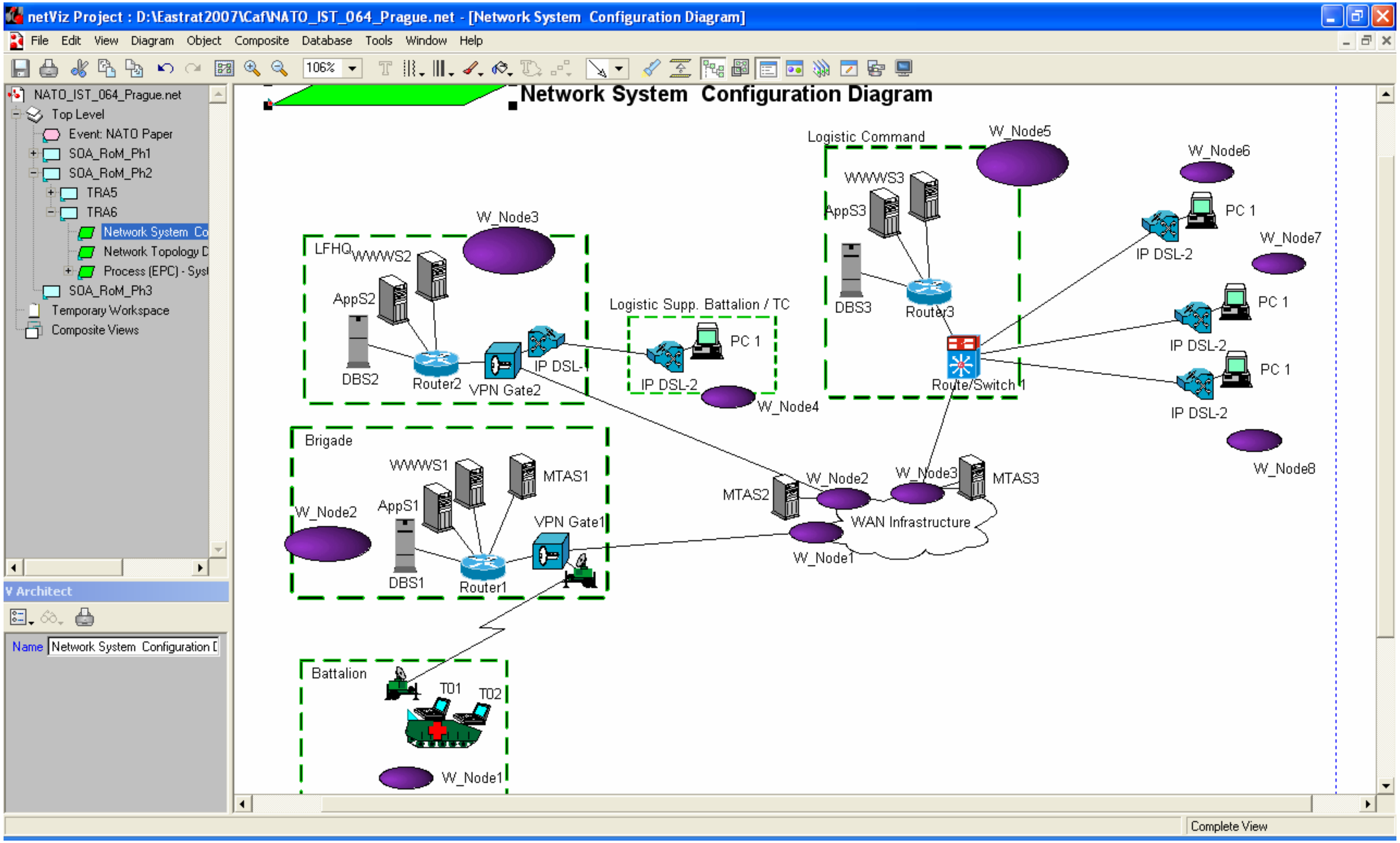
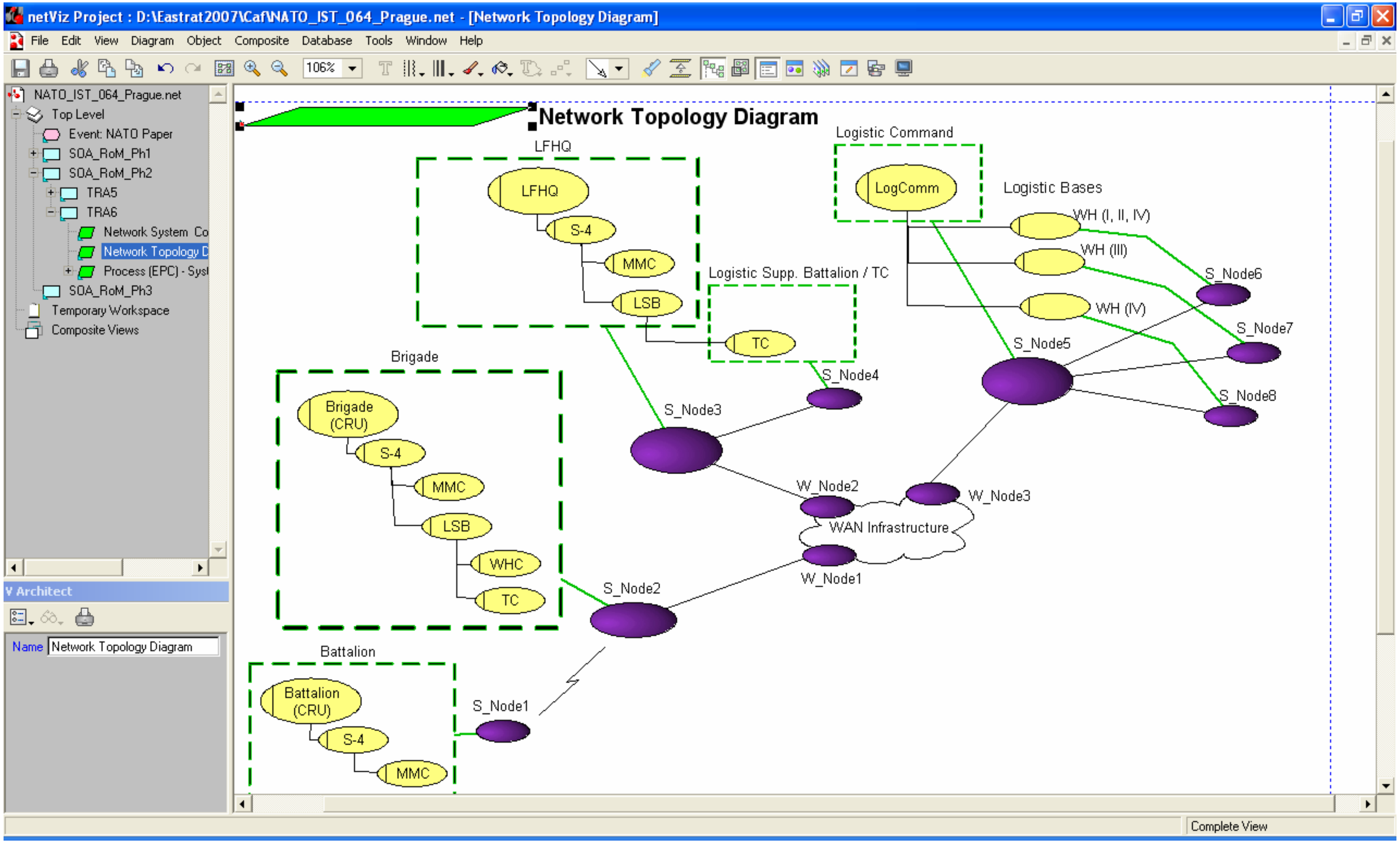
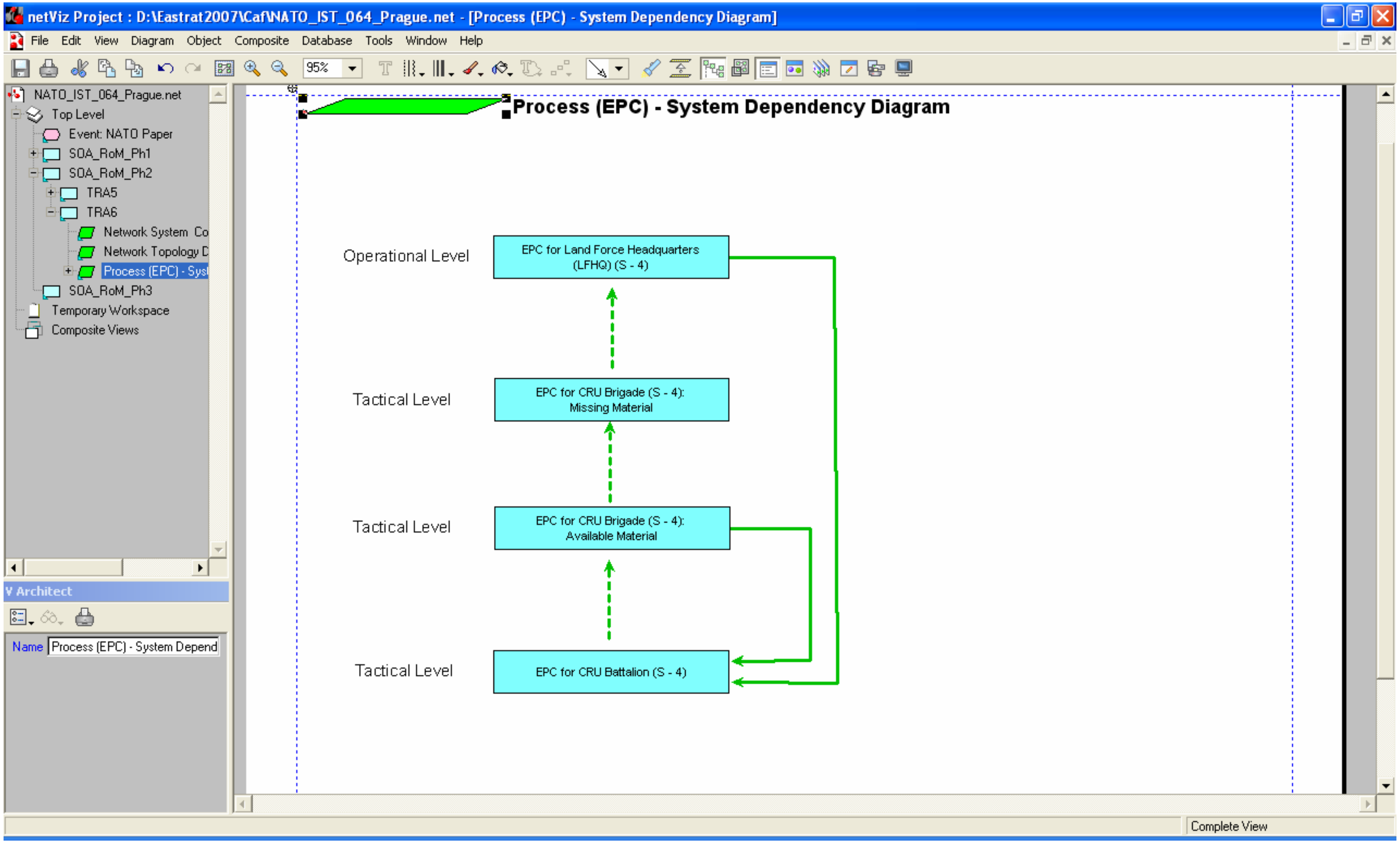


Figure 10: Network System Configuration Diagram Captures All ICT Components that Must be Available for the MRO Processing. ICT Components are Associated with Network Nodes.



netViz Project : D:\Eastrat2007\Caf\NATO_IST_064_Prague.net - [Network Topology Diagram]

Figure 11: Network Topology Diagram Captures All Organization Units Involved in the MRO Processing. Organization units are associated with network nodes.

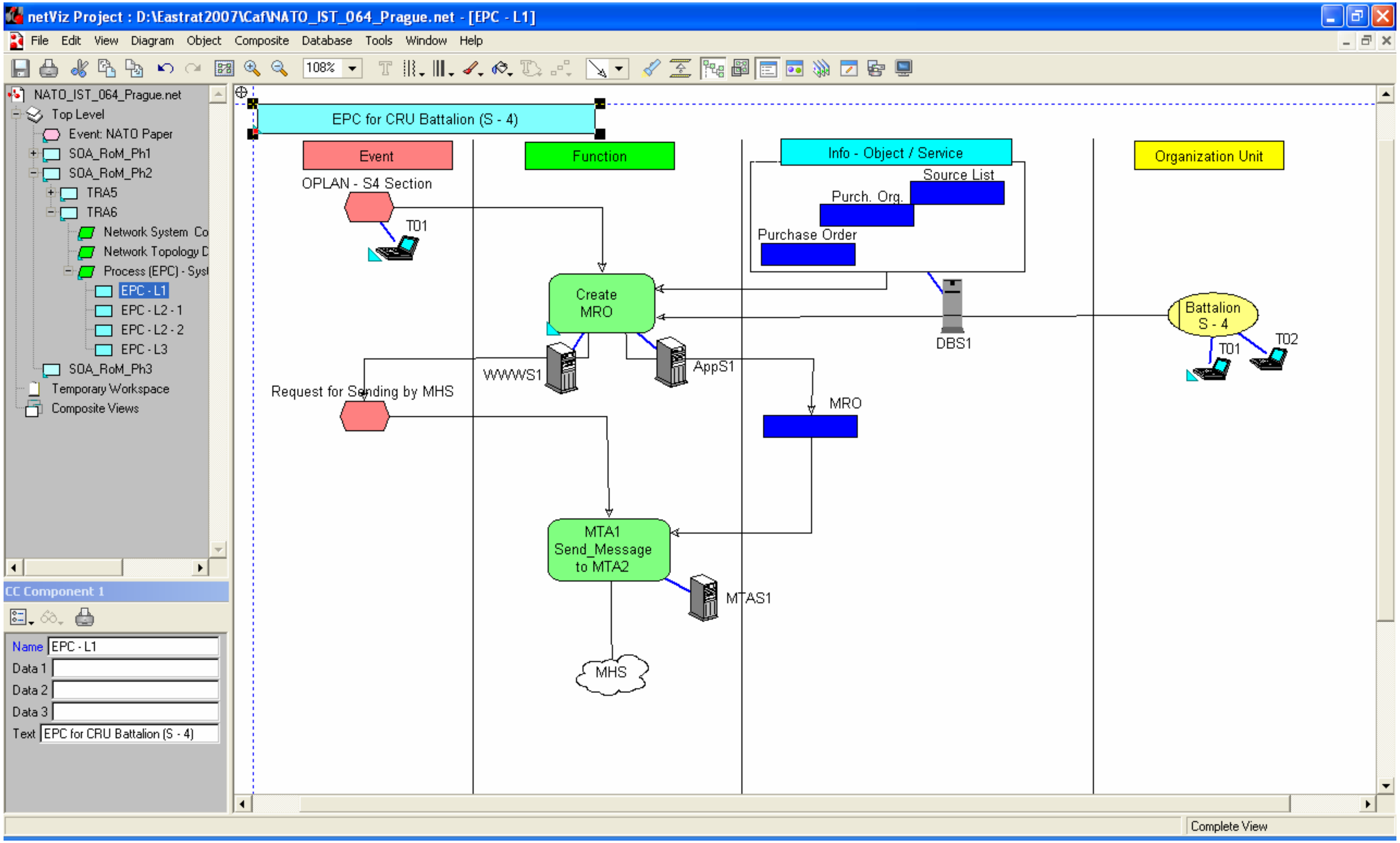

netViz Project : D:\Eastrat2007\Caf\NATO_IST_064_Prague.net - [Process (EPC) - System Dependency Diagram]

Name: Process (EPC) - System Depend

Process (EPC) - System Dependency Diagram

Operational Level
EPC for Land Force Headquarters (LFHQ) (S - 4)
Tactical Level
EPC for CRU Brigade (S - 4): Missing Material
Tactical Level
EPC for CRU Brigade (S - 4): Available Material
Tactical Level
EPC for CRU Battalion (S - 4)

Figure 12: Process (EPC) – System Dependency Diagram Copies EPC Diagram Used for Visualization of the MRO Processing (see Figure 5).



Name EPC - L1

Data 1

Data 2

Data 3

Text EPC for CRU Battalion (S - 4)

Figure 13: Process (EPC) – System Dependency Diagram is Applied for Each Organizational Level Involved in the MRO Processing (Figure 11). It allows understanding of dependencies among events, functions, and information objects (services) and ICT components.

5.0 – Minutes of the NATO RTO Workshop on “Building Robust Systems from Fallible Construction”

Prague, 9 and 10 November 2006

Prepared by Yves van de Vijver

5.1 AGENDA

Thursday, 9 November

9:00 Welcome, logistics, introductions of participants
- Morven Gentleman and Milan Snajder

9:15 Introduction to NATO Research and Technology Organization, and the Information Systems Technology Panel
- Lt. Col. Patrick Prodhome, IST Panel Executive

9:30 Introduction to topic and process
- Morven Gentleman

9:45 First working session: framing the problem – what are the challenges today and why is the research done in the past not enough?

- Systems of systems
- Pre-existing third-party code, including COTS and Open Source
- Installation and configuration errors
- Misunderstood interfaces and protocols
- Invalid operator input
- Malicious attacks
- Rate of upgrades
- …

10:30 Break

10:45 First working session resumes

12:30 Lunch

14:00 First working session resumes

15:30 Break

15:45 First working session resumes

16:30 Summary of day’s discussions

17:00 End of first day

Friday, 10 November

9:00 Second working session: what new technology or new approaches might reduce exposure risk or facilitate damage recovery

- Education to ingrain fault sensitivity awareness
- Architecture, especially Service Oriented Architecture
- Internet and Web technologies, especially web services
- Virtual machines
- Genetic programming
- Trouble sensors
- Situational awareness and autonomic frameworks
- Forward rather than rollback error recovery
- Dependability of adaptive systems
- …

10:30 Break

10:45 Second working session resumes

12:30 Lunch

14:00 Second working session resumes

15:30 Break

15:45 Way forward: Summary of conclusions and next steps

17:00 End of workshop

5.2 DAY 1, 9 NOVEMBER 2006

5.2.1 Welcome and Objective

The chairman of the task group and the workshop, Dr. Morven Gentleman (Canada), opens the meeting and welcomes the attendees to the workshop. The objective of the task group is to produce a report for the NATO community on the subject of robust systems/software development. The topic of robustness/ reliability is not new, but goes back all the way to the early days of computers and software. Therefore, the types of questions to be addressed in this workshop are the following:

- What has changed in the world of fault-tolerance and building systems in the last 30 years?
- What is different now compared to then?
- What has not been addressed before?
- What new techniques for building systems exist, and how do they deal with reliability?
- ….

After this welcome and introduction to the objective of the workshop, the NATO/RTO National Coordinator of the Czech Republic, <Name>, welcomes the attendees to the beautiful city of Prague and the magnificent workshop location at the Czech Ministry of Defence.

Next, Lt. Col. Patrick Prodhome, the IST Panel Executive, introduces NATO/RTO and the IST Panel. He shows the central position of the RTO in the NATO Structure and explains how the RTO works.

5.2.2 First Working Session

The first working session consisted of a number of short introductions describing the problem area and the invited experts' views on this area, followed by longer, in-depth presentations of the work of some of these experts.

5.2.2.1 Short Introductions and Views

Dr. Morven Gentleman

The first working session starts with a problem definition by the chairman of the task group, Dr. Morven Gentleman. He demonstrates the problem by means of a Geographical Information System (GIS), for instance an internet site with a map of the city of Prague, with public buildings, tourist attractions, public transport, etc., which you may ask for routes from A to B and the time it takes to get there. These systems usually contain static data, such as maps, roads and rivers, which change infrequently, and dynamic data, such as public transport schedules, and road works. More sophisticated systems may also take time-based data and weather into account (e.g. to calculate time to move from A to B). Often, this information is not in the system itself, but retrieved from external sources. And these external sources are systems of their own, developed and maintained by third-parties not under control of the GIS operations. Also, the usage of these external sources may not be as originally foreseen by the developers of that system. Finally, a main source of errors is wrong operator input, and this can happen anywhere in the system itself, or any of the external sources used.

The above example shows the many things that can go wrong when building an application on the basis of a collection of (external) components of which the reliability is unknown, or questionable. Building such an application from scratch, however, is an expensive solution, and hardly ever done (especially not in commercial applications).

The classical approaches towards reliability mostly duplicated components (different implementations) and used a voting system to determine which "answer" was the correct one, or used an "oracle" to decide whether an answer was ok, or not. A basic assumption in many of the classical approaches was the presence of an ability to roll-back to a previous (correct) state and try again from there. In current applications, such as the one above, this is not always possible. So what to do then?

This last question is also very relevant to the NATO context, in which coalitions have to be formed in a very short time frame, and in which the composition of coalitions change regularly. Each of the partners brings their own systems and these have to be put together quickly.

Prof. Alexander Romanovsky

Prof. Romanovsky (University of Newcastle) is an expert in (software) fault tolerance and explains that a shift has taken place in this field. Traditionally, software fault tolerance was built into systems to deal with hardware failures. But nowadays, faults are introduced because of architectural mismatches between different components or subsystems within the system, or because of faults in the underlying infrastructure in which the components rely. These different types of faults ask for different types of techniques to deal with them. And to add further to the problem, also in implementing fault tolerance, faults are introduced. Therefore, he would like to direct research towards "software engineering for fault tolerance", i.e. a method for software engineering in which fault tolerance is an inherit part of the method, not an add-on afterwards.

Prof. Mary Shaw

Prof. Shaw (Carnegie-Mellon University) is an expert in software architecture models. Recently, she has been researching high-level system design methods in which costs (e.g. for high assurance) are taken into account as an integral part of the design. The needs of the user play an important role in this. If, for example, the GIS application described by Dr. Gentleman earlier will be used by an ambulance service, reliability and accuracy must be much better than if the application will be used by tourists exploring the city. Same application, but a need for different designs (resulting in different costs).

Prof. Shaw also mentions an important difference between system design thirty years ago, and system design now: lack of central control. Where large systems in the past were usually designed by one chief designer, or a small team of designers, who had control over the whole system design, large systems today are more and more composed out of distributed components under distributed control. This introduces the problem that any of the components of the system may change, without notice, at any time.

Tomas Feglar MSc PhD

Tomas Feglar is an international consultant in Information Systems Research and Architecture and an expert in process integration and systems engineering. He has a large experience in working with the Czech Ministry of Defence. He notices that the eastern European countries and their defence organisations are modernizing and professionalizing fast. This challenge is accompanied by a rapid rate of change in Information Technology. He has been involved in meeting these challenges for the Czech Army and separated the two types of challenges by introducing business process models. First, a technology independent set of business models were composed for Army Force Management and Force Development. Then, these models are populated with technology and risks and threats to this technology are identified. In the systems engineering activity, these risks and threats are explicitly taken into account in the architectural design process, risk management process, information management process, and security management process.

As a final remark, Mr. Feglar would like to see robustness become a major topic in Service Oriented Architectures.

Maarten Boasson

Maarten Boasson is an expert in software architectures, especially for distributed applications. He worked for a major Dutch defence company and work on distributed, real-time architectures for, among others, frigates of The Royal Netherlands Navy. He always believed architecture was the answer to all problems, but over the years he has become more sceptical. When comparing software engineering with other engineering disciplines, he notices that there is no software engineering "discipline": no discipline, no theory, no guidelines. "We are just trying to build systems".

Most efforts within software engineering are addressed to produce systems with "very few" faults, but one fault may be too much. Most of the efforts in fault-tolerance are attempts to build a way around faults, but who are we fooling, but ourselves.

Robust systems must contain no faults. And in order to ever get there, software engineering should become a "real" engineering discipline.

Christophe Dony

Christophe Dony is an expert on exception handling. His current research consists of analysing languages for the construction of systems (object oriented, component based, service oriented architectures), and defining a new one, which will include exception handling mechanisms. Such mechanisms already exist to some degree in software languages (e.g. Java), but not for distributed systems, component based systems, etc.

Frédéric Painchaud M. Sc.

Frédéric Painchaud works for the Defence Research and Development Canada and is relative new to the area. He started researching this field because a client needs a framework for building fault-tolerant distributed systems. The short term goal of the research is to produce an overview of the available products for such a framework. The long term goal is to develop a new framework. Frédéric has kindly offered the results of his research so far for a state-of-the-art section in the task group's final report, for which the task group is very grateful.

Dr. Morven Gentleman – View of Experts not Present

The chairman of the task group, Dr. Morven Gentleman, finishes this first working session by shortly introducing some views of experts who could unfortunately not attend the meeting in person.

The first view considers autonomic computing in the form of self adapting or self healing software a solution to the problem of fault-tolerance. The software monitors itself and, in case of bad performance, adjusts the built-in control loops. Each individual component does this for itself, and the system for the system as a whole. This solution is considered most applicable in systems where "faults" do not occur instantaneously, but do occur as a result of degradation of performance over time.

The second view introduces recovery oriented design as a solution. Given the fact that errors will always be there (e.g. human operator input errors), design the system in such a way that it can always go back to a previous, safe state.

The third view abandons the concept of predictive engineering (think what can go wrong and build something to deal with that) and promotes to run a system in many different situations to find the faults that (may) occur, for instance by applying genetic algorithms. This view also promotes no to go back to previous safe states, but go to a stable state from where to continue. Of course, the problem is: "which are the stable states?"

5.2.2.2 Presentations

After the short introductions, the first working session was resumed with longer presentations, some of which were based on the position papers submitted.

Tomas Feglar MSc PhD – "SOA Robustness Roadmap"

Tomas identifies four important models to be developed for a system with a Service Oriented Architecture (SOA):

- SOA Robustness Decomposition;
- Enterprise Integration Model;
- Enterprise Application Model; and
- Decision Support Model.

In this presentation, he focuses on the first of these models. The system is decomposed in business processes, which provide one or more services to other business processes. System characteristics such as agility and ability are covered within the business process tier of the (layered) architecture. Reliability is covered within the SOA tier of the architecture.

For each of the business processes, risk profiles are defined in which the various threats and impacts are identified. These threats may be technical (e.g. hardware failure), but also physical (e.g. destruction of parts of the infrastructure because of war-time activity). For each risk identified, risk measures are determined. These measures may be measures for fault-tolerance (e.g. in case of hardware failure), but also management or logistic measures (e.g. spare parts).

The idea behind taking risk management into account from the beginning is that future changes to the system will be more economical, resulting in lower Total Ownership Costs.

During the work performed for the Czech Army, Tomas has discovered recurring patterns in these risk profiles and measures, and identified a number of "SOA Enterprise Robustness Patterns" which are used in numerous places throughout the overall system description.

The modelling activities are supported by a software tool, in which both the process view and the design view are maintained and synchronized. Tomas demonstrates the use of the tool taking an example from the position papers of F. Michaud and F. Painchaud.

Prof. Mary Shaw – "Strategies for Achieving Dependability in Coalitions of Systems"

Mary starts her presentation with trends in systems development and use: from local to distributed, from independent to interdependent, from insular to vulnerable, from installations to communities, from central administered to user managed, from software to resource, from single systems to coalitions. As a result of these trends, failures will ripple through these "systems of systems".

The dependability issues with these systems of systems are numerous:

- Different types of users require different levels of dependability.
- Costs matter (money, delay, disruption), but few can afford high dependability.
- Uncertainty is inevitable because specifications will never be complete, and actual operational environments are unknown.
- Integration is a bigger challenge than components.

In order to structure the problem domain and the discussions in the rest of the meeting, Mary shows a classification of types of measures for reliability. The first distinction is the time at which a problem will be considered: before using the system (preventive), or during use of the system (reactive). Preventive measures "validate" the system against a standard. This may be a global standard, a relative standard, or a policy standard. Validation against a global standard is the "traditional" approach, based on analysis and careful development. Other measures in this category are formal methods, and testing. Validation against relative standards take the user needs into account, and is therefore categorized as "User-centred". Validation against policy standards is an approach in the case of large systems of systems, in which negotiations among stakeholders drive the standard. This category is therefore called "Ultra-large scale". Reactive measures, or "remediation", can be further decomposed in technological reactive measures ("fault-tolerance"), technological adaptive measures ("self-adaptive, self-healing"), and economical reactive measures ("compensation").

The preventive measures ("validation") are common in many engineering disciplines. A common factor in these disciplines is the existence of linear models, which may be validated easily. But, in case of systems of systems, such linear models probably do not exist. Therefore, validation will be at least very difficult, if not impossible. As a consequence, for systems of systems, reactive strategies to dependability will probably be more effective.

Maarten Boasson – "Software Faults"

Maarten identifies tree types of faults: faults that lead to no results; faults that lead to wrong results; and faults that lead to late results. Faults are the result of errors. In order to prevent faults, first of all, errors must be detected before they occur. Secondly, the impact of errors must be limited, and, thirdly, errors must be repaired. Maarten would like to see that effort and research is focused on the first way of prevention, error detection, or, even better, error prevention ("never stop striving for correct programs"). He therefore is a strong advocate for research into formal data models, formal verification, and minimal dependencies (e.g. by late binding, asynchronous communication, and autonomous components).

Prof. Joe Sventek – "Closed-Loop Management Patterns"

Joe presents the idea of using closed-loop management patterns in system management. Closed-loop management is used in many production plants: measuring the output of a process, and adjusting the input and control of the process to steer the process towards producing the desired output. This principle works well for managing processes with slowly degrading output. So why not adopting such an approach to managing systems and software processes?

To make the use of closed-loop management even more powerful, the system to be managed could be designed as a hierarchy of lower-level processes or subsystems. On each of these levels, and in each of the components or subsystems at those levels, the closed-loop management principle can be applied. The advantages are manifold, including the closure of a loop as low as possible in the hierarchy will avoid the propagation of problems, and some well-proven techniques at one level can be reused at other levels.

The open question in applying this principle to achieve robustness, is how well this principle will work for faults, i.e. not a slowly degradation of performance, but a sudden malfunctioning of the system?

Prof. Cristophe Dony

Cristophe's interests are agent-based and component-based applications and message oriented programming. The objective of his current research is to specify and implement an Exception Handling System (EHS) for new architectures, for instance, J2EE.

Cristophe defines an exceptional situation as a situation in which the standard execution cannot continue. An Exception Handling System must then raise the exception, associate handlers to entities to deal with the exception, and put back the system to a coherent state.

Current solutions towards an EHS include:

1) Standard signalling methods (e.g. Java), which are stack-oriented and destructive;
2) Separation of treatment; and
3) Contextualization (pass exception to requester who knows the context).

These solutions, however, do not work well when dealing with concurrency, which is inevitable in today's larger and larger, distributed systems. Commercial implementations of exception handling systems for these types of systems are not available yet. But, Prof. Dony et al. have been able to produce a first implementation of an exception handling system for J2EE.

He is currently looking to coordinate his activities with other researchers in the field.

Frédéric Painchaud

Frédéric introduces the research he has been performing in the last eighteen months on fault-tolerance. He identifies three sources of faults: the system environment, the hardware, and the software. Most error recovery strategies can be classified as either: retry, fallback (redundancy), or diversity (different implementations of the same algorithm). Most of the times, these strategies fail, because the underlying logical problem is not recognized, and similar mistakes are made independently of each other.

He started researching this field because a client needs a framework for building fault-tolerant distributed systems. The short term goal of the research was to produce an overview of the available products for such a framework. The long term goal is to develop a new framework.

The anticipated solution is a framework for systematic development of software with built-in support for managing diversity, based on the principles of Erlang (Ericsson), and developed for JAVA with support for distributed applications.

5.3 DAY 2, 10 NOVEMBER 2006

5.3.1 Second Working Session – Presentations

Maarten Boasson – SPLICE

The second working session starts with a presentation of the ideas behind SPLICE, a dependable, distributed architecture developed by Maarten Boasson during his activities at Hollandse Signaal Apparaten BV. A commercial implementation (OpenSplice) is available.

The basic idea behind the architecture is that a system works when "the right information is at the right place at the right time". The design principles behind SPLICE are: make inter-process data visible, minimize dependencies, and maximize component autonomy. The resulting architecture is deceptively simple, but very powerful.

SPLICE contains a number of mechanisms for fault-tolerance: passive replication ("cold-start"), semi-active replication ("hot-start"), and active replication. For robustness purposes, all messages should pass absolute states, not relative states, in order to allow components to send the same message multiple times without changing the meaning (e.g., turn left 10 degrees cannot be sent more than once, because if two messages are received, the ship would turn 20 degrees; a message "at time T, change course to 70°" may be sent as many times as needed to assure one message is received).

After this presentation, a number of questions were raised by the audience:

Q1: Is there a notion of reflection?

A1: Yes, processes can look at system data and determine which processes are subscribed to what, which processes are running, etc.

Q2: Is deadlock/ live-lock used as a programming tool as it is often in blackboard systems?

A2: Deadlock/Live-lock never occurred, so the situation is not explicitly dealt with. Starvation may happen, but deadlock not.

Q3: How to ensure real-time behaviour?

A3: Hard real-time is not practical, so just assure empirically, or by calculation, that messages are passed quickly enough for proper working of the system. Usually, passing time-stamped information is enough to make a successful hard real-time system. Most hard real-time requirements are not really hard. Usually it is some sort of periodic calculation.

Q4: Some data passed is large ("blobs") and fixed length messages are not very useful, which means you have to send sequences of messages. How is this done in SPLICE?

A4: There are two types of messages in SPLICE. The first type is "fire and forget", for which length is irrelevant. The second type is "order maintained", for which messages are chopped up, sent, and assembled as an atomic action before giving the message to the subscriber.

Q5: How do you establish that a collection of processes together are a semantically correct system? Is there a chief designer?

A5: In practice, there has always been one "group" of designers with control. But the architecture provides some support in the form of "watchdog processes", which monitor the running and dying processes. Furthermore, the publish/subscribe mechanism allows publishers without subscribers, and there may even be multiple publishers, but this is not recommended. Crucial for the working of the system, is to get all required data structures right.

The discussion finally led to the following problem statement by Mary Shaw that would guide the remainder of the working session:

> "If you compose your system by using externally developed components or systems (e.g., a database available through the internet), how do you assure your overall application is correct?"

This is a specifically new challenge since this way of composing systems is relatively new. And in the area of Service Oriented Architectures (SOAs), which aim to meet this challenge, research into robustness of SOAs is not yet covered (very well).

With this problem statement, presentations by attending experts have finished, and the meeting moves to a smaller round-table room in order to facilitate interactive discussions better.

5.3.2 Second Working Session – Round Table Discussion

The chairman of the meeting, Morven Gentleman, starts the round table discussions with some more ideas on what is different now compared to thirty years ago, and how these differences may be used to improve robustness. One major difference is computing power and network capacity, which was once scarce but now often in excess. Why not use the excess power for speculative computing (e.g. data mining) on archived data such as event logs and incident logs to discover something has gone wrong or might go wrong? Add "audit routines" to the software that, based on earlier results, suggest that something is not working correctly. In the example of the GIS application at the start of the meeting (beginning of Section 5.2.2.1), structured data could be audited to get an idea of the quality of the data. Also, consistency with unstructured data could be checked (is a road in the structured data visible on the satellite image?). But finding inconsistencies is one side of the problem; the other side is what to do if one is found? How do you notify the running application that something's wrong, and what is the application supposed to do then?

Mary Shaw suggests that such an approach should be coupled with a model of confidence for the application and raise flags, but maybe should not directly impact the running application. Morven Gentleman replies that there should be grades in audit routines, such that serious problems are dealt with immediately, and minor problems are left to deal with later, unless more audit routines raise the same issue, and the "problem level" is increased.

Patrick Prodhome – Guidelines for Final Report

Because some attendees will have to leave the meeting earlier, the round-table discussion is shortly adjourned to enable Patrick Prodhome, NATO RTO IST Panel Executive, to provide the attendees with some guidelines for contributing to the final report of the task group. An important prerequisite is the submission of a publication clearance through the proper national NATO channels, in order for RTO to be able to publish the final report. Patrick also shows how to log onto the RTO web site and what materials may be found there.

Round Table Discussion Continued

The discussion continues on the use of the spare capacity in the military environment. Where commercially "just-in-time" seems to be the prevailing attitude (which usually turns out to be "just-too-late"), military systems are specifically designed to handle peak loads (war-time). When not needed (peace-time), spare capacity can be used to do training exercises, or simulations, to ensure that the system will work when it is really needed. This way of using resources is part of the military culture, where personnel are used in a similar way. Therefore, military systems are good candidates to apply the suggestions of Morven Gentleman to.

Cristophe Dony remarks that adding more constructs to a system for robustness usually tends to obscure the structure of the system and make the system more difficult to understand. Alexander Romanovsky follows up on that remark by stating that the contrary might be the case if constructs are used that allow separating normal computation from exceptions. In this case, the understanding of the system may even be increased. This view is supported by Tomas Feglar, who uses risk management strategies to build robustness into systems top down. He claims the resulting system is better, needs less repairing during operational lifetime, and therefore the initial structure is better maintained.

After these comments, the discussion is again focussed on the question: What is different now, compared to thirty years ago? Mary Shaw has been looking into "Ultra Large Scale Systems" recently, and found the following list of differences:

- Decentralised operation and control.
- Conflicting, unknowable, diverse requirements.
- Continuous evolution and deployment.
- Heterogeneous, inconsistent, and changing elements.
- Indistinct people/system boundary.
- Normal failures.
- New forms of acquisition and policy.

After lunch, Morven Gentleman, re-starts the discussion from a slightly different angle. In the seventies, techniques of redundant data structures, which allowed to check damage in data, and sometimes even allowed automated repair, were popular. Have these techniques been forgotten, or are they still around? Alexander Romanovsky knows that diversity in data structures, either by a mapping function, or by double implementation, is still used to recover data structures. Mary Shaw reminds that in user interfaces, checkable redundancy (e.g. address and postal code) is sometimes built in to check consistency of manual input. But, in general, it may be worthwhile to look into some of these "old" techniques, and see what has become of them, to include in the final report. There may even be techniques that once where abandoned because of, for instance, lack of computing power, and that could be revived because of the changed conditions in computing.

Alexander Romanovsky follows up this discussion with the observation that there seem to be many diverse implementations of the same functionality on the Internet. Maybe this is a new form of redundancy that could be exploited for fault tolerance? Morven Gentleman, however, remarks that, although the interface (web page) is different, the underlying implementation could still be the same, and that there is less diversity than you expect. Mary Shaw supports that view by adding that most weather sites are using the same underlying weather data from a single source. Alexander counters that, even in this case, the location of the site may be important for speed, and depending on your application, this could be a very important requirement.

The discussion is wrapped-up with some final thoughts on how all of the techniques and issues discussed in the previous two days will in the end better support the commander in asymmetric warfare; because that is the kind of question the NATO and its member states would like to see answered in the final report. The answer to this question, borrowed from Maarten Boasson in his presentation of SPLICE, is that these techniques will eventually help to get:

> "The right information, at the right place, at the right time."

5.4 MEETING CLOSURE

The meeting is closed with some ideas on how to produce the Task Group's Final Report:

- Morven Gentleman will, with the help of Yves van de Vijver, produce and circulate an outline to the meeting attendees and task group members, who are requested to give feedback and submit missing items.
- The possibility of a next meeting, for instance in June 2007, to actually work on the report will be investigated.
- Presentations of this workshop will be included in the Final Report as an appendix. Authors are requested to get publication approval through their national channels in order to enable the inclusion.

Finally, Morven Gentleman, as chairman of the task group and the meeting, thanks the attendees for their enthusiasm and valuable contributions to the meeting and the task group's activities, and the local organizing committee for hosting the meeting in such an inspiring environment.

And, finally, Milan Snajder, task group member and part of the local organizing committee, surprises the attendees by some nice gifts to remember the workshop and the visit to the beautiful city of Prague.

6.0 – Future Work

W. Morven Gentleman

Morven.Gentleman@dal.ca

This workshop has highlighted a number of take-away actions for the scientific community interested in software system dependability.

A final report describing the results of the workshop was produced and made available to others interested in the topic. The report was published in April 2008 as RTO-TR-IST-047, "Building Robust Systems with Fallible Construction". It details research that has yet to be done, but that has been identified as needed in order for systems being developed today to be resilient to predictable problems.

Existing software fault tolerance technology is inadequate because of new perspectives on what is it means to build robust systems, and what is needed for systems to be robust:

- Robustness is needed, which is a different issue from correctness.
- People are part of the system.
- Dependability requirements depend on which stakeholder is considered.
- Automated correction of failures is not always feasible or appropriate.
- Autonomic computing, i.e. self-managed systems, has a role.
- Rollback is not always feasible or desirable.
- Service availability may outweigh correctness of individual service requests.
- Software development is not a single homogeneous activity.
- The software product may not be monolithic homogeneous code.
- The development organization may not be a monolithic homogeneous entity.
- Malicious attacks may be an essential concern, beyond accidental failures.
- Fault tolerance awareness needs to be ingrained in stakeholders.

These points are elaborated in the aforementioned report. Recovery-oriented computing has also been recognized as an important shift of perspective, and although some investigations have been undertaken based it, nevertheless a great deal more is needed.

Existing software fault tolerance technology is also inadequate because new technology creates new options, but in addition poses new situations requiring additional solutions.

We have identified four technologies that have become widely available in the past few years, each of which offers potential for new solutions to building more robust systems:

1) Surfeit of computing capacity.
2) Autonomic computing.
3) Virtual machines.
4) The discipline of software architecture.

The aforementioned report elaborates on these technologies. Because these technologies were not available when most software fault tolerance technology was being developed, their application was not taken into account in the software fault tolerance literature. The potential benefits of them need to be investigated.

We have also identified about a dozen technologies that have become prominent in the past few years that present new challenges for building robust systems:

- Software component-based engineering.
- Systems of Systems.
- Web and Internet technologies.
- Concurrent, parallel, and distributed computing.
- Exception handling.
- Non-imperative programming.
- Genetic and more generally exploratory computation.
- Massive datasets.
- Inadequacy of oracles.
- Security and privacy.
- Multimedia, especially time-based streaming media.
- Scalability and nonstop operation.
- Rapid rate of new releases.

Again the aforementioned report elaborates on these technologies and issues that they raise. Although some of the existing literature on software fault tolerance bears on these issues, it has many deficiencies and gaps, meaning that intense further study is required in order to provide guidance for developing systems that involve these technologies.

In short, existing software fault tolerance literature has serious shortcomings for today's systems, and simple extension of past work will not resolve that: new directions in research must be pursued. Results are needed for systems being built now. Funding such research is critical.

Chapter 7 – Conclusions and Recommendations

W. Morven Gentleman

Morven.Gentleman@dal.ca

“Sunny-day” systems, that fail to take into account predictable things that might cause the system to fail, are inadequate and all too common. Ignoring consequences of fallible construction is a perfect example. Understanding of software dependability needs to be ingrained in all stakeholders, not just system designers:

- Robustness needs to be built-in, and cannot be an add-on.
- Implementation faults (bugs) are only one concern, maybe not even the most critical.

Software dependability cannot be achieved by middleware alone. Robust components are nice but also not enough.

Systems are made up of hardware, software, and people. Processes (i.e. workflow processes, not just the operating system software process abstraction) are critical, as are tools to support the processes. In the event of failure, systems need to be able to continue to operate, albeit in degraded mode, and ultimately to return to full operations status: graceful degradation epitomizes robustness. Procurement needs to take a much more pro-active stance on requiring robustness and recovery. Operations needs to plan for, and practice, recovery from failures. The inevitability of human failure needs to be recognized and planned for, not just in the form of end-user invalid input and misinterpretation of output, but also in incorrect actions of operational staff as well as support such as system installers and maintainers. Particularly insidious are data corruption faults that do not immediately give indication of failure, but may lurk undetected for days, weeks, even years. Ultra-large scale systems have highlighted problems that traditional approaches don’t solve. Nevertheless, some of these problems have long existed even in smaller systems, such as systems-of-systems.

Recovery-Oriented Computing is different from Disaster Recovery. The latter is typically taken into account by operational units as addressing business continuity and business resumption in the face of cataclysmic, but exceedingly unlikely events such as national power or communications outages, or aircraft crashing into computer centers or in the military context severe battlefield reverses. These vulnerabilities are indeed wise to consider, but are unlikely to warrant investment in automated response. Recovery-Oriented Computing, on the other hand, investigates automated aid to assist operational staff in restoring service after unavoidable events for which the risk is enough that the investment is prudent.

Although research on software fault tolerance has been done for almost 40 years, more needs to be done. This is definitely not polishing a shiny jewel, but rather addressing problems not yet resolved, problems of direct relevance to today’s net-centric systems. Funding must be found for this research.

The original plans for this workshop proved overambitious. Significant results were achieved, so this is not a criticism in and of itself, but it stresses the need for continuing efforts.

REPORT DOCUMENTATION PAGE

1. Recipient's Reference	2. Originator's References	3. Further Reference	4. Security Classification of Document
	RTO-MP-IST-064 AC/323(IST-064)TP/256	ISBN 978-92-837-0081-4	UNCLASSIFIED/ UNLIMITED

5. Originator

Research and Technology Organisation
North Atlantic Treaty Organisation
BP 25, F-92201 Neuilly-sur-Seine Cedex, France

6. Title

Building Robust Systems with Fallible Construction

7. Presented at/Sponsored by

This Report documents the material presented at the IST-064/RWS-011 Workshop held in Prague, Czech Republic, 9-10 November 2006.

8. Author(s)/Editor(s)	9. Date
Multiple	May 2009

10. Author's/Editor's Address	11. Pages
Multiple	118

12. Distribution Statement

There are no restrictions on the distribution of this document. Information about the availability of this and other RTO unclassified publications is given on the back cover.

13. Keywords/Descriptors

Commercial equipment
Computer applications
Computer architecture
Computer programs
Correlation
Critical system
Design
Distributed systems
Failure
Fault tolerance
Integrated systems
International cooperation
Interoperability
Methodology
Reliability
Software development
Software engineering
Software reuse
Standards
System of systems
Systems analysis
Systems engineering

14. Abstract

This workshop is related to Software Fault Tolerance, a topic that has been studied at least since 1970. Since then much has been learned about how to address those problems, as they were then understood. However changes in perspective as to what constitute the challenges, and changes in available and commonplace technology, have led to a need to go beyond conclusions reached in the past. The workshop was organized to review past and present understanding of the challenge, as well as examining relevant approaches to address them. Rather than an exchange of pre-prepared material, the workshop was intended as a working meeting with a goal of producing a deliverable that is a summary of the state of the art. The proceedings include position statements from the participants, slides from the presentations made by the participants, and the one complete paper that was submitted. Minutes of the discussions provide insight into how the deliverable, the final report of task group IST-047/RTG-019, was shaped.

NORTH ATLANTIC TREATY ORGANISATION

RESEARCH AND TECHNOLOGY ORGANISATION

BP 25
F-92201 NEUILLY-SUR-SEINE CEDEX • FRANCE
Télécopie 0(1)55.61.22.99 • E-mail mailbox@rta.nato.int

DIFFUSION DES PUBLICATIONS
RTO NON CLASSIFIEES

Les publications de l'AGARD et de la RTO peuvent parfois être obtenues auprès des centres nationaux de distribution indiqués ci-dessous. Si vous souhaitez recevoir toutes les publications de la RTO, ou simplement celles qui concernent certains Panels, vous pouvez demander d'être inclus soit à titre personnel, soit au nom de votre organisation, sur la liste d'envoi.

Les publications de la RTO et de l'AGARD sont également en vente auprès des agences de vente indiquées ci-dessous.

Les demandes de documents RTO ou AGARD doivent comporter la dénomination « RTO » ou « AGARD » selon le cas, suivi du numéro de série. Des informations analogues, telles que le titre est la date de publication sont souhaitables.

Si vous souhaitez recevoir une notification électronique de la disponibilité des rapports de la RTO au fur et à mesure de leur publication, vous pouvez consulter notre site Web (www.rto.nato.int) et vous abonner à ce service.

CENTRES DE DIFFUSION NATIONAUX

ALLEMAGNE
Streitkräfteamt / Abteilung III
Fachinformationszentrum der Bundeswehr (FIZBw)
Gorch-Fock-Straße 7, D-53229 Bonn

BELGIQUE
Royal High Institute for Defence – KHID/IRSD/RHID
Management of Scientific & Technological Research
for Defence, National RTO Coordinator
Royal Military Academy – Campus Renaissance
Renaissancelaan 30, 1000 Bruxelles

CANADA
DSIGRD2 – Bibliothécaire des ressources du savoir
R et D pour la défense Canada
Ministère de la Défense nationale
305, rue Rideau, 9e étage
Ottawa, Ontario K1A 0K2

DANEMARK
Danish Acquisition and Logistics Organization (DALO)
Lautrupbjerg 1-5, 2750 Ballerup

ESPAGNE
SDG TECEN / DGAM
C/ Arturo Soria 289
Madrid 28033

ETATS-UNIS
NASA Center for AeroSpace Information (CASI)
7115 Standard Drive
Hanover, MD 21076-1320

FRANCE
O.N.E.R.A. (ISP)
29, Avenue de la Division Leclerc
BP 72, 92322 Châtillon Cedex

GRECE (Correspondant)
Defence Industry & Research General
Directorate, Research Directorate
Fakinos Base Camp, S.T.G. 1020
Holargos, Athens

HONGRIE
Department for Scientific Analysis
Institute of Military Technology
Ministry of Defence
P O Box 26
H-1525 Budapest

ITALIE
General Secretariat of Defence and
National Armaments Directorate
5th Department – Technological
Research
Via XX Settembre 123
00187 Roma

LUXEMBOURG
Voir Belgique

NORVEGE
Norwegian Defence Research
Establishment
Attn: Biblioteket
P.O. Box 25
NO-2007 Kjeller

PAYS-BAS
Royal Netherlands Military
Academy Library
P.O. Box 90.002
4800 PA Breda

POLOGNE
Centralny Ośrodek Naukowej
Informacji Wojskowej
Al. Jerozolimskie 97
00-909 Warszawa

PORTUGAL
Estado Maior da Força Aérea
SDFA – Centro de Documentação
Alfragide
P-2720 Amadora

REPUBLIQUE TCHEQUE
LOM PRAHA s. p.
o. z. VTÚLaPVO
Mladoboleslavská 944
PO Box 18
197 21 Praha 9

ROUMANIE
Romanian National Distribution
Centre
Armaments Department
9-11, Drumul Taberei Street
Sector 6
061353, Bucharest

ROYAUME-UNI
Dstl Knowledge and Information
Services
Building 247
Porton Down
Salisbury SP4 0JQ

SLOVAQUIE
Akadémia ozbrojených síl
M.R. Štefánika, Distribučné a
informačné stredisko RTO
Demanova 393, P.O.Box 45
031 19 Liptovský Mikuláš

SLOVENIE
Ministry of Defence
Central Registry for EU and
NATO
Vojkova 55
1000 Ljubljana

TURQUIE
Milli Savunma Bakanlığı (MSB)
ARGE ve Teknoloji Dairesi
Başkanlığı
06650 Bakanliklar
Ankara

AGENCES DE VENTE

**NASA Center for AeroSpace
Information (CASI)**
7115 Standard Drive
Hanover, MD 21076-1320
ETATS-UNIS

**The British Library Document
Supply Centre**
Boston Spa, Wetherby
West Yorkshire LS23 7BQ
ROYAUME-UNI

**Canada Institute for Scientific and
Technical Information (CISTI)**
National Research Council Acquisitions
Montreal Road, Building M-55
Ottawa K1A 0S2, CANADA

Les demandes de documents RTO ou AGARD doivent comporter la dénomination « RTO » ou « AGARD » selon le cas, suivie du numéro de série (par exemple AGARD-AG-315). Des informations analogues, telles que le titre et la date de publication sont souhaitables. Des références bibliographiques complètes ainsi que des résumés des publications RTO et AGARD figurent dans les journaux suivants :

Scientific and Technical Aerospace Reports (STAR)
STAR peut être consulté en ligne au localisateur de ressources
uniformes (URL) suivant: http://www.sti.nasa.gov/Pubs/star/Star.html
STAR est édité par CASI dans le cadre du programme
NASA d'information scientifique et technique (STI)
STI Program Office, MS 157A
NASA Langley Research Center
Hampton, Virginia 23681-0001
ETATS-UNIS

Government Reports Announcements & Index (GRA&I)
publié par le National Technical Information Service
Springfield
Virginia 2216
ETATS-UNIS
(accessible également en mode interactif dans la base de
données bibliographiques en ligne du NTIS, et sur CD-ROM)

NORTH ATLANTIC TREATY ORGANISATION

RESEARCH AND TECHNOLOGY ORGANISATION

BP 25
F-92201 NEUILLY-SUR-SEINE CEDEX • FRANCE
Télécopie 0(1)55.61.22.99 • E-mail mailbox@rta.nato.int

DISTRIBUTION OF UNCLASSIFIED RTO PUBLICATIONS

AGARD & RTO publications are sometimes available from the National Distribution Centres listed below. If you wish to receive all RTO reports, or just those relating to one or more specific RTO Panels, they may be willing to include you (or your Organisation) in their distribution.

RTO and AGARD reports may also be purchased from the Sales Agencies listed below.

Requests for RTO or AGARD documents should include the word 'RTO' or 'AGARD', as appropriate, followed by the serial number. Collateral information such as title and publication date is desirable.

If you wish to receive electronic notification of RTO reports as they are published, please visit our website (www.rto.nato.int) from where you can register for this service.

NATIONAL DISTRIBUTION CENTRES

BELGIUM
Royal High Institute for Defence – KHID/IRSD/RHID
Management of Scientific & Technological Research
for Defence, National RTO Coordinator
Royal Military Academy – Campus Renaissance
Renaissancelaan 30
1000 Brussels

CANADA
DRDKIM2 – Knowledge Resources Librarian
Defence R&D Canada
Department of National Defence
305 Rideau Street, 9th Floor
Ottawa, Ontario K1A 0K2

CZECH REPUBLIC
LOM PRAHA s. p.
o. z. VTÚLaPVO
Mladoboleslavská 944
PO Box 18
197 21 Praha 9

DENMARK
Danish Acquisition and Logistics Organization (DALO)
Lautrupbjerg 1-5
2750 Ballerup

FRANCE
O.N.E.R.A. (ISP)
29, Avenue de la Division Leclerc
BP 72, 92322 Châtillon Cedex

GERMANY
Streitkräfteamt / Abteilung III
Fachinformationszentrum der Bundeswehr (FIZBw)
Gorch-Fock-Straße 7
D-53229 Bonn

GREECE (Point of Contact)
Defence Industry & Research General Directorate
Research Directorate, Fakinos Base Camp
S.T.G. 1020
Holargos, Athens

HUNGARY
Department for Scientific Analysis
Institute of Military Technology
Ministry of Defence
P O Box 26
H-1525 Budapest

ITALY
General Secretariat of Defence and
National Armaments Directorate
5th Department – Technological
Research
Via XX Settembre 123
00187 Roma

LUXEMBOURG
See Belgium

NETHERLANDS
Royal Netherlands Military
Academy Library
P.O. Box 90.002
4800 PA Breda

NORWAY
Norwegian Defence Research
Establishment
Attn: Biblioteket
P.O. Box 25
NO-2007 Kjeller

POLAND
Centralny Ośrodek Naukowej
Informacji Wojskowej
Al. Jerozolimskie 97
00-909 Warszawa

PORTUGAL
Estado Maior da Força Aérea
SDFA – Centro de Documentação
Alfragide
P-2720 Amadora

ROMANIA
Romanian National Distribution
Centre
Armaments Department
9-11, Drumul Taberei Street
Sector 6, 061353, Bucharest

SLOVAKIA
Akadémia ozbrojených síl
M.R. Štefánika, Distribučné a
informačné stredisko RTO
Demanova 393, P.O.Box 45
031 19 Liptovský Mikuláš

SLOVENIA
Ministry of Defence
Central Registry for EU & NATO
Vojkova 55
1000 Ljubljana

SPAIN
SDG TECEN / DGAM
C/ Arturo Soria 289
Madrid 28033

TURKEY
Milli Savunma Bakanlığı (MSB)
ARGE ve Teknoloji Dairesi
Başkanlığı
06650 Bakanliklar – Ankara

UNITED KINGDOM
Dstl Knowledge and Information
Services
Building 247
Porton Down
Salisbury SP4 0JQ

UNITED STATES
NASA Center for AeroSpace
Information (CASI)
7115 Standard Drive
Hanover, MD 21076-1320

SALES AGENCIES

**NASA Center for AeroSpace
Information (CASI)**
7115 Standard Drive
Hanover, MD 21076-1320
UNITED STATES

**The British Library Document
Supply Centre**
Boston Spa, Wetherby
West Yorkshire LS23 7BQ
UNITED KINGDOM

**Canada Institute for Scientific and
Technical Information (CISTI)**
National Research Council Acquisitions
Montreal Road, Building M-55
Ottawa K1A 0S2, CANADA

Requests for RTO or AGARD documents should include the word 'RTO' or 'AGARD', as appropriate, followed by the serial number (for example AGARD-AG-315). Collateral information such as title and publication date is desirable. Full bibliographical references and abstracts of RTO and AGARD publications are given in the following journals:

Scientific and Technical Aerospace Reports (STAR)
STAR is available on-line at the following uniform resource
locator: http://www.sti.nasa.gov/Pubs/star/Star.html
STAR is published by CASI for the NASA Scientific
and Technical Information (STI) Program
STI Program Office, MS 157A
NASA Langley Research Center
Hampton, Virginia 23681-0001
UNITED STATES

Government Reports Announcements & Index (GRA&I)
published by the National Technical Information Service
Springfield
Virginia 2216
UNITED STATES
(also available online in the NTIS Bibliographic Database
or on CD-ROM)

ISBN 978-92-837-0081-4