An Integrated Architecture Analysis Framework for Component-Based Software Development

Novia Admodisastro

PhD in Computing

Computing Department

Lancaster University

A thesis submitted for the degree of Doctor of Philosophy

October 2011

Declaration

I certify that this thesis submitted for the degree of PhD in Computing is the result of my own research, except where acknowledged, and that this thesis or any part of it has not been submitted for higher degree to any other university or institution.

Signed : .....

NOVIA INDRIATY ADMODISASTRO

Date : 28/10/2011

Abstract

The importance of architecture in reuse-driven development is widely recognized. Architecture provides a framework for establishing a match between available components and the system context. It is a key part of the system documentation; enforces the integrity of component composition and provides a basis for managing change. However, one of the most difficult problems in component-based system development (CBD) is ensuring that the software architecture provides an acceptable match with its intended application, business and evolutionary context. Unlike custom development where architectural design relies solely on a detailed requirements specification and where deficiencies in application context can be corrected by 'tweaking' the source code, in component-based system development the typical unit of development is often a black-box component whose source code is inaccessible to the developer. Getting the architecture right is therefore key to ensuring quality in a component-based system. Architecture analysis in CBD provides the developer with a means to expose interface mismatches, assess configurations with respect to specific structural and behavioural constraints and to verify the adequacy of compositions with respect to quality constraints. However, support for key component-based system design issues is still patchy in most architecture analysis approaches. My solution has been to develop, *Component-based Software Architecture analysis FramEwork*

(CSAFE), a scenario-driven architecture analysis approach that combines and extends the strengths of current approaches using pluggable analysis. CSAFE is process-pluggable and recognises that negotiation (trade-off analysis) is central to black-box software development. However, while CSAFE is primarily intended to support black-box development, we recognise that there may be aspects of the system for which a black-box solution is not feasible. CSAFE supports custom development in such situations by treating abstract components as placeholders for custom development. CSAFE is supported by an extensible toolset.

Acknowledgements

In the Name of Allah, the Beneficent, the Merciful. All praise is due to Allah, whose Bounties and Mercies I cannot begin to enumerate. Not in my wildest dream, I could complete this journey, when time is so bleak and dim hope, and only losses keep accompany, to Allah I thankful for giving me strength to carry on.

My most deepest and earnest appreciation goes to my supervisor, Dr. Gerald Kotonya, his enthusiasm, his inspiration, and his guidance on this journey has been invaluable. Tirelessly, he provided encouragement, sound advice, good teaching, and lots of good ideas. I could not ask for a better supervisor than him, thank you for believing in me.

I am also most indebted to the Universiti Putra Malaysia for granting me the study leave and to the Government of Malaysia for awarding me a generous sponsorship. Sincere thanks to Lancaster University which has granted me William Ritchie Travel Fund during the study period to attend a prestige international conference in Beijing, China.

My colleagues and administration staffs from the Lancaster Department of Computing for supporting me in my research work. I want to thank them for all their help and support.

The affection and heart-warming friendship I received from miraculous people who have I known in Lancaster University, to name a few Laila Alabadi, Azrina Kamaruddin, Rafidah Md Noor, Xiaozhu Wu, Sara Khan, Christine Dawson, cannot be described, only deeply felt.

This thesis is devoted to my parent, Wargo Admodisastro and Mainah Macartney, for their previous love and support in all my efforts, and for giving me the foundation to be who I am. To my beloved father, rest in peace, you will always be my greatest admirable guide and philosopher. To my baby sister Vienna, who gives me comfort with her cheerful stories, keep on telling good stories.

There have been many others who have give encouragements and prayers and to all I am grateful.

To all the good memories...

"So verily, with hardship, there is relief."(94:5)

List of Figures

List of Tables

Publications

Admodisastro, N. and Kotonya, G.: *Usability Requirements for Architectural Analysis Tool to Support CBD*. In: Proc. of the 2nd International Conference on User Science and Engineering (i-USer). IEEE Computer Society, 2011; 118-123.

Admodisastro, N. and Kotonya, G.: An Architecture Analysis Approach for Supporting Black-box Software Development. In: Crnkovic, I., Gruhn, V. and Book, M. (ed.) ECSA 2011. *LNCS*, vol. 6903: 180-189, Springer, Heidelberg, 2011.

Admodisastro, N. and Kotonya, G.: *An Architectural Analysis Approach for Black-box Component-Based Systems*. In: Proceeding of the 2nd Annual GSTF International Conference on Software Engineering (SE), 2010; 68-74 – *Awarded for Best Research Student Paper*

Admodisastro, N. and Kotonya, G.: Architectural Analysis Approaches: A Component-Based System Development Perspective. In: Mei, H. (ed.) ICSR 2008. *LNCS*, vol. 5030: 26-38, Springer, Heidelberg, 2008.

Admodisastro, N. and Kotonya, G.: *Towards an Integrated Approach to Architectural Analysis in Component-based Software Development*. Poster in London Hopper

Colloquium of British Computer Society (BCS). 2007. London, United Kingdom.

Admodisastro, N. and Kotonya, G.: *An Integrated Approach to Architectural Analysis in Component-based Software Development*. Poster in Christmas Conference of Faculty of Science & Technology. 2006. Lancaster University, Lancaster.

Admodisastro, N. and Kotonya, G.: *Towards an Integrated Approach to Architectural Analysis in Component-based Software Development*, In: Proceeding of the Work in Progress Session in the 32nd IEEE EuroMicro Conference (SEAA), Dubrovnik, Croatia. 2006.

Table of Contents

Chapter 1

Introduction

Component-Based Software Engineering (CBSE) is a sub-discipline of software engineering. However, it derives its motivation from traditional engineering. Britannica encyclopaedia [Britannica10] defines engineering as a profession that is devoted to designing, constructing, and operating the structure, machines, and other devices of industry and everyday life. An important characteristic of traditional engineering rarely builds whole systems from scratch; instead traditional engineering has established time-tested principles to construct systems from pre-fabricated parts. For instance, automotive engineering designs and produces cars by fitting together basic components such as window, door, fender, engine etc. through an assembly line process [Auto10]. This is an essential characteristic of a mature engineering discipline. The main advantage of this is that high quality systems can be produced more cheaply and rapidly than custom-built systems since standard reusable components can be massively produced and tested in different user contexts.

Component-based software system development typifies traditional engineering philosophy by promoting the construction of systems from pre-fabricated software components. Underlying this philosophy is the promise of accelerated, low cost

development and reliable software systems [Clements10]. However, the use of components to build software systems is not a new idea. Software "componentization" was first proposed in 1969 by Doug Mcllory [Wiki10] as a way of tackling the "software crisis". Similarly, according to Jacobson [Heineman01], more than 30 years ago components called "blocks" were used to construct a telecommunications system. The system became the largest commercial software success story in Sweden and inspired many software engineers to use blocks to construct all kinds of applications. In recent years CBSE has been transformed into a practical software development approach by the emergence of commercial component technology and standards.

CBSE offers a different approach to the conventional software reuse in that it encompasses architecture [Shaw96], design patterns [Gamma95], component frameworks [Fayad97] and is constrained by the availability of suitable third party components. In summary:

- Component-based design is a negotiated process that is subject not only to user requirements, but also to the availability of suitable off-the-shelf software components.
- A component is integrated with other components and/or frameworks via a plug-and-play mechanism, thus components can be composed at run-time without compilation. Component-based development is interface-centric. Hiding the implementation part allows components to be composed without the need to know their internal details.
- Components are associated with particular component models and frameworks. This means that a component requires standardization of its interface.
- Component can be acquired via market distribution and improved in quality through market competition [Aoyama98].

Fig. 1.1 illustrates the different types of reusable development elements in CBSE.

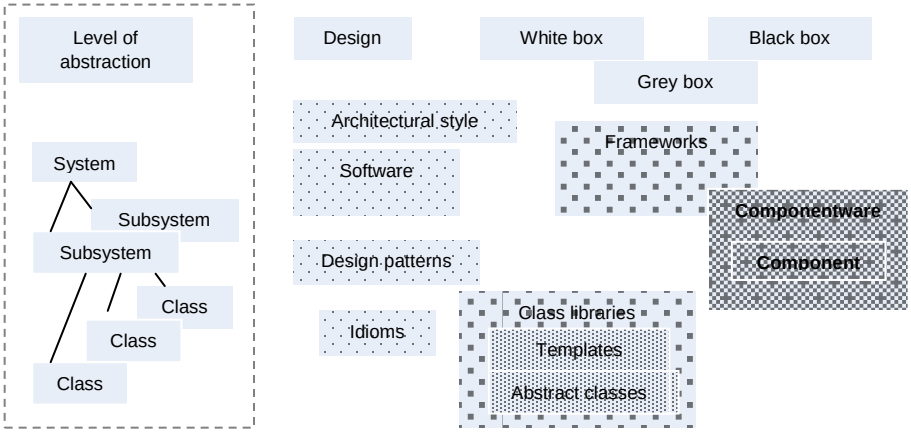


Fig. 1.1 Components reside in software reuse [Aoyama97]

1.1 CBSE in Practice

The software industry has increasingly adopted a component-based approach to software development. The software community has realized that CBSE can provide tremendous benefit if harnessed properly. According to Feblowitz and Greenspan [Feblowitz98], organisations developing software are turning to software components in the hope of reducing the risks associated with software development. The underlying factors behind the increasing use of software components include:

- Improved reliability.
- The possibility of attaining shorter time-to-market for products. Simply because the buying organisations do not need to make everything from scratch.
- The quality of the component is regarded as being higher after undergone extensive process reuse by customers, possibly even competitors. It is reported that the reuse process allows the management to expect substantial gains, time to market: reductions of 2 to 5 times, defect density: reductions of 5 to 10 times, maintenance cost: reductions of 5 to 10 times, and overall software development cost: reduction of around 15% to as much as 75% [Jacobson97].

Organisations that have adopted CBSE have reported similar benefits. Bond et al. [Bond05] describe their experience of implementing a distributed feature composition framework (DFC) for a large international telephone company. The DFC is component-based architecture for the development of complex telecommunication services. The DFC architecture is designed to provide feature modularity and structured feature composition. In the DFC, a request for service is satisfied by a dynamically assembled graph of concurrent processes implementing feature functions and a point-to-point connection. Bond et. al. describe their experience and the result as extremely rewarding and a clear demonstration of the value of CBSE in telecommunications. Upadhyaya [Upadhyaya08] describes a successful experience of developing of a large component-based application that handles massive federal and a United States state government labor market information data.

ABB, a global power and automation company, used a component-based approach to develop the Open Control System (OCS) [Advant10], a large embedded product-line system designed to suit different industrial applications that include systems for power utilities, power plants and infrastructure, and the petroleum industry. The National Electric Company of Japan (NEC) used component-based software development to construct the HolonEnterprise [Aoyama01], a large distributed store management and point-of-sales (POS) system to support the NEC's chain of stores across Japan. Other CBSE success stories are published in [Luer01].

However, despite these relative successes, component-based software engineering is still hampered by the lack of practical methods and tools that support the reuse-driven paradigm embodied in black-box components. Some of the key challenges are discussed next.

1.2 Challenges for Developing Systems from Components

Software components represent an attempt to exploit the advantages of genuinely reusable software. Components promise potentially greater rewards because packaged expertise can be purchased in an open market place. However, the limited visibility of black-box components and the variability in application contexts means that the specifications delivered with third party components are often incomplete or inadequate. This in turn means that the correspondence between stakeholder concerns and the system architecture, and the correspondence between the system architecture and components is often unclear. Broadly, component-based system development poses seven challenges:

- *Component discovery and verification.* Off-the-shelf software components have to be discovered, understood and, sometimes adapted to work in a new environment. For the development process to be successful, it must provide mechanisms for discovering, verifying, adapting and 'wiring' plug-compatible components.
- *Balancing need and availability.* There is a conceptual gap between the way we articulate requirements in custom development and the reuse-driven paradigm embodied in black-box component-based system development. The features supported by commercial software solutions vary greatly in quality and complexity. This together with the variability in application contexts means that specifications delivered with black-box software are likely to be inadequate [Vidger96].
- *Architecting the system.* A typical component-based system architecture comprises a set of components that have been purposefully designed and structured to ensure that they fit together and have an acceptable match with a defined system context. However, poor support for negotiation and lack of effective techniques for defining, verifying, evolving and matching abstract designs to concrete components make this a difficult task.

- *Supporting diversity*. The increasing complexity and diversity of software systems means that it is unlikely that large systems will continue to be developed using a purely component-oriented approach. Rather, a hybrid model of software development is likely to emerge where components and other solutions such as web services co-exist in the same system.
- *Managing change*. Traditional system maintenance involves observing and modifying lines of code. However, in component-based development the main unit of construction is often a black-box component or service. This limited visibility to the component design presents fundamentally different change management tasks and has major implications for the way we manage and evolve composition-based systems [Kotonya05a].
- *Poor standard descriptions*. There are several modelling notations intended to support component-based development. Perhaps the best known is the Unified Modelling Language (UML) [Pilone05]. However, while the recent versions of UML offer some support with constructs for modelling component-based systems, these are largely intended to support custom development (UML does not support the notion of component discovery and verification). UML component diagrams are not intended to provide a logical decomposition of a software system into reusable and combinable subsystems. In addition, UML modelling is largely domain-driven, which usually leads to designs based on domain objects and non-standard architectures. Lastly, UML provides no easy way of addressing "compositional mismatches".
- *Poor tool support*. Component-based development environments are typified by tools such as WREN [Luer01], model driven approaches such as ASF+SDF [van den Brand01] and component tools for Networked Embedded Systems (NEST) [Volgyesi02]. Many of these include the ability to locate potential components from component distribution sites and to incorporate selected components into application design models. However, they provide little support requirements formulation, negotiation, architecture analysis, design pattern reuse, or "glue-code"

generation, and no support for managing change. Model driven initiatives are largely domain-specific and intended for developing reusable components, rather than systems from pre-existing components.

The work described in this thesis is addresses challenges of developing viable architectures for component-based systems.

1.3 Motivation for Research

Component-based System Development (CBD) focuses on the realization of systems through integration of pre-existing components [Bass05,Crnkovic02,Medvidovic07]. A component is reusable software element that exposes its functionalities (services) through one or more interface and, can be independently deployed and composed without modification [Heineman01]. There is also the possibility to acquire software components from third parties, commonly known as Off-The-Shelf (OTS) components [Li08] (e.g., commercial OTS components or COTS; free components open source or FOSS [Feller02]; and web services [Papazoglou08]). Software architecture plays an important role in CBD as it provides a framework for establishing a match between available components and the system context. Architecture contributes not only to the system documentation, it contributes to the integrity of the component composition, maintenance, and evolution. However, one of the most difficult problems in CBD is ensuring that the software architecture provides an acceptable match with its intended application, business and evolutionary context [Medvidovic07].

Unlike custom development where architectural design relies solely on detailed requirements specification and where deficiencies in application context can be corrected by 'tweaking' the source code, in CBD the typical unit of development is often a black-box component whose source code is inaccessible to the developer. Unfortunately, features supported by third party software components often vary greatly in quality and complexity. In addition, the contexts in which the components

are used may also vary considerably. This complexity together with the variability in application contexts means that the documentation supplied with software components is often incomplete or inadequate. Additional analysis is often required to ensure that an acceptable solution is achieved, and to address situations where unforeseen user needs coincide with a component's undocumented design assumptions. Architecture analysis can provide an effective and relatively low-cost mechanism for addressing these problems.

Architecture analysis can provide means to expose interface mismatches, assess configurations with respect to specific structural and behavioural constraints and to verify the adequacy of compositions with respect to the application context. Architecture analysis can also provide a basis for developing "what-if" scenarios to explore the implications of evolving a component-based system [Kotonya05a,Dobrica02]. However, current architecture analysis approaches differ widely with respect to their underlying models, analytical capabilities and ability to support CBD making it difficult for developers to ascertain their effectiveness in different application contexts [Hutchinson05,Abowd97]. Current architecture analysis schemes vary from process embedded models that derive skeleton architectures by matching non-functional requirements to architectural styles [Wallnau03], to stakeholder-driven schemes that analyze architectures using multiple quality attributes to identify and improve areas of highest risk [Kazman98], to aspect-oriented approaches that use cross-cutting system properties to suggest improvements to system architecture [Viera00].

A key challenge in developing black-box software systems is how to provide developers with tools that allow them to derive suitable software architectures by balancing aspects of stakeholder concerns with the architectural considerations and capabilities embodied in software components. It is important to note that a component-based system architecture is both an expression of required functionality

and the result of verifying the suitability of the components used. Getting the architecture right, therefore, has a major impact on the quality of the final system.

This thesis describes CSAFE, a scenario-driven architecture analysis approach that provides a framework for balancing aspects of stakeholder concerns with architectural considerations and component solutions to derive viable system architectures. CSAFE is process-pluggable to minimise process disruption and supports the analysis of different architectural aspects.

1.4 Objectives

The aim of this research was to develop a pluggable architecture analysis framework for component-based systems that integrated and extended the strengths of current approaches. The framework was primarily intended for black-box development, but would allow white box development in situations where black-box development was not feasible. In summary the objectives of the research were:

1. To formulate a classification and comparison framework that could be used to assess the efficacy of software architecture analysis approaches in black-box development.
2. To use (1) to develop a scenario-driven architecture analysis framework to support black-box component-based development. In addition to supporting the requirements in (1), the framework should:
   (i) Allow the system designer to adapt and tailor the design process to reflect the system context and domain specific needs (i.e. be process-pluggable).
   (ii) Provide explicit support for broad stakeholder involvement.
   (iii) Provide support for pluggable architecture analysis.
   (iv) Provide explicit support for trade-off analysis (i.e. negotiation).
   (v) Provide support for standard design notations.
3. To develop an extensible toolset to support the architecture analysis framework
4. To evaluate the framework on non-trivial case studies.

1.5 Research Contributions

The contributions of this research are as follows:

1. The first contribution of this research is the formulation of a classification and comparison framework for software architecture analysis approaches [Admodisastro08]. The framework consists of eight key requirements that can be used to design architectural analysis methods and assess their efficacy for component-based development.
2. The second contribution is the development of *Component-based Software Architectural Analysis Framework (CSAFE)*, a scenario-driven, negotiation-based architecture analysis framework for black-box component-based software development [Admodisastro06].
3. The third contribution is development of an extensible toolset to support CSAFE. The toolset supports diversity in analysis by supporting pluggable analysis that allow different tools to be incorporated. The toolset also supports an extensible XML repository of design templates and components that allows the system designer to define analysis contexts that include design patterns, styles and organisation-specific schemes.
4. The fourth contribution is the development of UML parser and the iXML architecture description language to support the transformation and verification of UML and iXML architecture descriptions. The parser transforms UML architectures into processable specifications (i.e. iXML), and the ADL provides a mechanism of verifying the correctness of iXML architectures.
5. The fifth contribution is the results of evaluating CSAFE in static and runtime conditions. The first evaluation uses a real case study drawn from an Electronic Document and Delivery Interchange System (EDDIS) project to demonstrate CSAFE features and its practicability [Admodisastro10] and Guided Vehicle Parking System. The second evaluation uses runtime system behaviour to validate the efficacy of CSAFE architectural refinements.

1.6 Thesis Structure

The remainder of this thesis is organised as follows:

- Chapter 2 sets the context for the research by providing a background on software architecture and its relation to CBSE. The notion of software architecture and its relationship to system quality is discussed. The chapter then discusses the importance of software architecture evaluation. The chapter concludes with a discussion of current CBSE process models.
- Chapter 3 discusses design challenges in component-based system development. The chapter uses the design challenges to identify the necessary requirements for architecture analysis in CBD. The chapter then uses the requirements to assess existing architecture analysis approaches intended to support component-based development.
- Chapter 4 presents the proposed architecture analysis solution, Component-Based Software Architectural Analysis Framework (CSAFE). Various features of the framework including its underlying method and toolset are presented.
- Chapter 5 presents the first of two CSAFE evaluations. The first evaluation demonstrates the features of CSAFE and the practicability of the framework using a case study drawn from an electronic document delivery and interchange system project.
- Chapter 6 presents the second CSAFE evaluation. The second evaluation focuses on the runtime validation of architectural refinements.
- Finally, chapter 7 provides some concluding thoughts and further work.

Chapter 2

Background

Like custom-developed systems, component-based systems must continue to satisfy evolving stakeholder needs (i.e. stakeholders must remain confident that their concerns are addressed in the design solution) and the system must adapt to an ever-changing environment [Vidger01]. It is therefore crucial that system architects understand how stakeholder concerns are addressed in the application, how they relate to other system concerns and how changes in the system might affect them [Saniabille01].

In component-based software systems the basic building block is a black-box component, there is no code to act as "final documentation" of the system and any inadequacy in component documentation may represent "lost information". This limited visibility of components represents a real risk for system architects when it comes to tracking stakeholder concerns, understanding how well they are addressed in the system design. Systematic architectural analysis can help ensure that risks resulting from architectural adaptations and trade-offs do not adversely affect critical system attributes. The analysis is likely to reveal not only how well an architecture satisfies a

particular context, but also how changes to specific quality attribute might affect other quality concerns.

This section provides a background to software architecture. The relationship between architecture and system quality, and architecture evaluation is discussed. The section closes with a review of how the issue of system quality and architecture evaluation is addressed in three representative component-based system development models.

2.1 Software Architecture

The study of software architecture began in 1968 when Edsger Dijkstra pointed out that it pays to be concerned with how software is partitioned and structured, as opposed to simply programming [Clements96]. Today as software has become larger and more complex, that assertion has been proven. A software system must have a systematic representation that works as a blueprint to give the software engineer the "big picture" before system details are committed to implementation [Stafford01a].

Although software architecture is often regarded as a high-level description of the organization of a software system, it has been described in slightly different ways by researchers in software engineering. Perry and Wolf [Perry92] provide the classical definition of software architecture as a 3-tuple consisting of *elements, form and rationale*. Over the years other researchers have extended and refined this classical definition. Elements represent the systems building blocks (e.g. objects, components and services). Elements may be considered at different levels of abstraction to manage complexity and improve communication amongst system stakeholders. Form captures the ways in which the system elements are organised in the architecture. It represents the structure of individual architectural elements, and the manner in which they are composed in the system. Lastly, form characterises the interactions and relationships of the elements with their operating environment. Rationale represents the systems

designer's intent, assumptions, subtle choices, external constraints, selected design patterns and styles.

Shaw and Garlan [Shaw96] state that software architecture encompasses components (e.g. database, middleware and ports), connectors to enable communication, constraints to define how components can be integrated, and semantic models to understand its overall properties. According to Clements [Clements95] software architecture is a way to structure systems so that they can be built from reusable components. Clements underscores the importance of architecture in facilitating the component interconnection, rapid system evolution and reliable analysis.

Philippe Kruchten [Kruchten95] uses a model composed 5 views to define software architecture. The model is known the 4+1 architectural view model and comprises:

- the *logical view*, which is the object model of the design,
- the *process view*, which captures the concurrency and synchronization aspects of the design,
- the *physical view*, which describes the mapping(s) of the software onto the hardware and reflects its distributed aspect, the development view, which describes the static organization of the software in its development environment.

The description of an architecture, and design decisions, is organized around these four views, and then illustrated by a few selected use cases, or scenarios that become the fifth view. For each view the set of elements to be used is defined (i.e. components, containers and connectors). Each view is described by a blueprint using its own particular notation. For each view also, the architects can pick a certain architectural style, hence allowing the coexistence of multiple styles in one system.

Bass et al. [Bass05] define software architecture as:

"*The structure or structures of a program or computing system, which comprises software elements, the externally visible properties of those elements, and the relationships among them.*"

The importance of this definition is that it recognizes the need for multiple representations to describe the architecture of a single system. Each of these representations may have its own concept of elements and relationships. Furthermore, Bass views architecture as an inherent property of all systems, meaning that all systems have an architecture, even if it is not explicitly specified or even known. The work described in this thesis adopts the architecture definition by Bass et al. [Bass05].

Architecture is critical to system quality. The architecture of the software system can affect the system's availability, safety, performance, security, efficiency, robustness and maintainability. The particular style and structure chosen for a software system may therefore depend on non-functional system requirements (NFRs). Sommerville [Sommerville01] gives the example of a critical performance requirement that may influence the system architecture should be designed to localise critical operations within a small number of sub-system with as little communication as possible between the sub-systems. This may mean using relatively large-grain components to reduce component communications. On the other hand, if security were a critical requirement, a layered structure for the architecture may be preferred with the most critical assets protected in the innermost layers, and with a high level of security validation applied to these layers. More important is the fact that quality attributes are not mutually exclusive. The achievement of one quality attribute invariably impacts positively or negatively on other quality attributes.

2.2 Architecture and System Quality

Bass et al. [Bass05] state that,

"*Although functionality and other qualities are closely related, functionality often takes not only the front seat in the development scheme, but the only seat. This is short-sighted, however. Systems are frequently redesigned not because they are functionality deficient but because they are difficult to maintain, port, or scale, or too slow, or have been compromised by network hackers.*"

This is a common scenario, qualities are rarely taken into account in most software development processes. According to Clements [Clements95] these requirements are not explicitly dealt with because of their complexity, usually informal statement, high abstraction level, as well as the rare support of languages, methodologies, and tools for them. The problem is more difficult for non-trivial systems with competing quality requirements. For example, reliability and performance often exist in a state of mutual tension, data replication to increase reliability will decrease the performance of the system. The software engineer has to trade-off these attributes to achieve an acceptable level of system quality.

Achieving acceptable quality must be considered throughout the design, implementation and deployment of a system. There is no quality attribute that is entirely dependent on design, nor is it entirely dependent on implementation or deployment. Satisfactory results are a matter of getting the architecture as well as the implementation right. Therefore two conclusions can be made. Firstly architecture is critical to the realization of many qualities in a system, and these qualities should be designed in and be evaluated at the architectural level. Secondly, architecture by itself is unable to achieve complete system quality. However, it provides the foundation for achieving quality, but this foundation will be of no avail if attention is not paid to the system implementation.

Quality attributes can be classified in several ways. Bass et al. [Bass05] classify quality attributes into three main categories: qualities of the system, business qualities, and architectural qualities. Qualities of the system focus on considerations such as availability, modifiability, security, usability and safety. While business qualities are goals which frequently shape a system's architecture, they include cost, schedule, market, and marketing. Architectural qualities are directly related to the architecture itself namely conceptual integrity, correctness and completeness, and buildability. Architectural qualities represent the conceptual underlying theme that unifies the design of the system at all levels. Whereas, correctness and completeness are essential for the architecture to allow for all of the system's requirements and runtime resource constraints to be met, buildability allows the system to be completed by the available team in a timely manner and to be open to certain changes as its development progresses.

Therefore it is perhaps unsurprising that qualities can be classified according to a number of "views" or "perspectives" (see Fig. 2.1). Each view comes from a particular context (e.g. business analyst), any single view tends to give only a partial picture. The views identified tend to be stereotypical, as such a distinction is commonly made within software quality between the "user or client" and the "designer or supplier". The views are generally presented in adversarial pairs such as users versus designers. Satisfying the non-functional requirements (NFRs) which may be synergistic or conflicting requires a process of negotiation and trade-off.

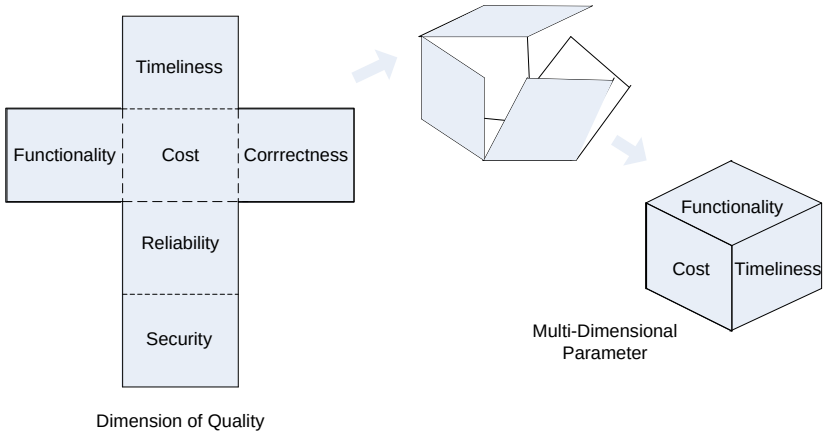


Fig. 2.1 Quality in multi-dimensional construct [Gillies96]

2.2.1 Achieving Qualities: Architectural Styles, Patterns, Custom, Metrics, and Scenarios

An architectural pattern in software, also known as an architectural style [Bass05], is analogous to an architectural style in building architecture, such as *Gothic* or *Greek Revival*. It consists of a few key features and rules for combining them so that architectural integrity is preserved. It is important to mention that not all researchers agree that architectural style is the same as architectural pattern. The disagreement is due to different level of granularity perceived by these researchers between architectural pattern and architectural style. Architectural style is considers as a coarse-grained pattern that provides an abstract design decisions for designing a software.

An architectural style is determined by:

- A set of elements types (e.g. data repository or a component that computes a mathematical function).
- A topological layout of the elements indicating their interrelationships.
- A set of semantic constraints (e.g. filters in a pipe-and-filter style are pure data transducers which incrementally transform their input stream into an output stream, but do not control either upstream or downstream elements).

- A set of interaction mechanisms (e.g. layered, blackboard, object-oriented) that determine the way elements are coordinated through the allowed topology.

A building architect, Christopher Alexander [Gamma95] once said, *"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice".*

According to Lüders [Lüders00] architectural styles defined the vocabulary of the design. Benefits of applying well-known or standardized architectural styles include possibilities of design and code reuse, ease of understanding the architecture, and increased interoperability. Having the same belief, Shaw and Garlan's [Shaw96] influential work attempted to catalogue a set of architectural patterns that are known as architectural styles. This is analogous to design patterns [Gamma95] and code pattern [Coplien97] at different level of abstraction.

The motivation for Shaw and Garlan's project was the observation that high-level abstraction for complex systems exist in software engineering as in other engineering disciplines. These patterns occur regularly in system designs, but without systematic cataloguing it prevents the software engineer from recognizing them, because in different disciplines the same architectural pattern may be called different things. Fig. 2.2 shows how patterns are categorized into related groups in an inheritance hierarchy. *Event systems,* for example, is a sub-style of independent elements, and has two sub-patterns: implicit invocation and explicit invocation.

Another study by Perry and Wolf [Perry92] suggested the use of architectural style for constraining the architecture and coordinating software architects. They proposed that rationale, together with elements and form, constitutes the model architecture. Perry and Wolf [Perry92] illustrated a number of interesting architectural points in building architecture that have corresponding mappings in software architecture. This

is particularly true of architectural styles like network, hardware, and web-engineering [Pressman09].

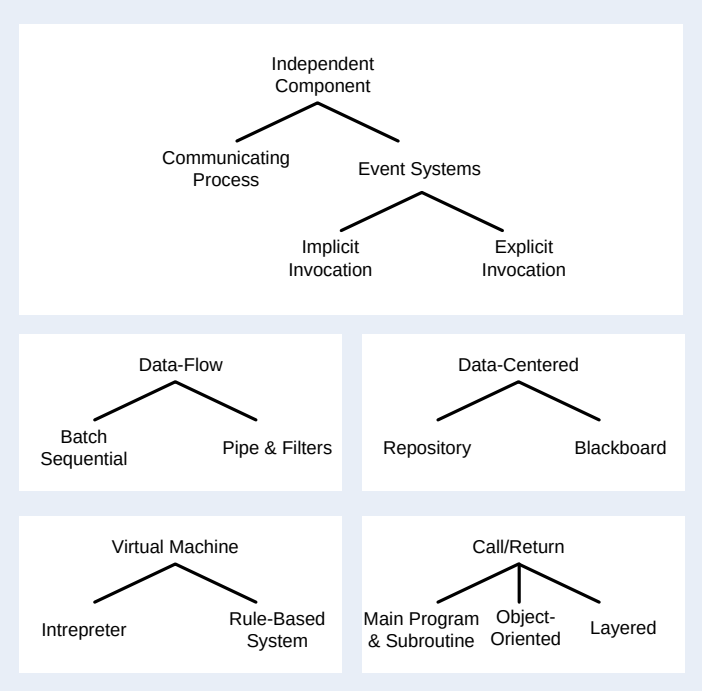


Fig. 2.2 A sample catalogue of architectural patterns, organised by is-a relationship [Bass05]

2.3 Software Architecture Evaluation

Systematic architecture analysis can help ensure that risks resulting from architectural adaptations and trade-offs do not adversely affect critical system quality attributes (e.g. performance, security and modifiability). The analysis is likely to reveal not only how well an architecture satisfies a particular quality attribute, but also how architectural changes to improve one quality attribute might affect other quality attributes. These decisions are likely to have a profound effect on the quality of the delivered system. The architecture analysis process can not only reveal how well an architecture satisfies particular quality attributes, it can also provide insight into how those attributes interact and the implications of trading them off against each other.

Some benefits that accrue from holding early architectural evaluation [Bass05] include:

1. *Validation of requirements*. Discussion and examination of how well an architecture meets requirements often opens up the requirements for discussion. Requirements creation, isolated from early design, usually results in conflicting system properties. High performance, security, fault tolerance, and low cost are all easy to demand, but difficult to achieve, and often impossible to achieve simultaneously. Architecture evaluation can uncover the conflict and trade-offs, and provide a forum for their negotiated resolution.
2. *Forces preparation for the review*. Requiring a representation before the evaluation is done means that reviewees must document the system's architecture. The process of preparing for the evaluation is likely to expose problems that the software architecture may have to address.
3. *Captured rationale*. A documented design rationale explaining the design choices and their rationale is a critical part of the software system life cycle. Hence the implications for design modifications can be assessed.
4. *Early detection of problem with the existing architecture*. The earlier in the life cycle problems are detected, the cheaper it is to fix them. The problems that can be found by an architecture evaluation include unreasonable (or expensive) requirement, performance problems, and problems associated with potential downstream modifications. In this way an architecture evaluation can provide early insight into software system capability and limitations.
5. *Improves architecture*. Organizations that practice architecture evaluation as a standard part of their development process report an improvement in the quality of the architecture that is evaluated. As development organizations learn to anticipate the questions that will be asked, the issues that will be raised, and the documentation that will be required for evaluation, they naturally pre-position themselves to maximize their performance on the evaluation. Architecture evaluation results in better architecture not only after the fact, but before the fact as

well. Over time, an organization develops a culture that promotes good architectural design.

6. *Cost savings*. A study of several large projects by AT&T reported that each project manager perceived savings from architecture evaluation. Over an eight-year period, projects receiving a full architecture evaluation reported an average 10% reduction in project costs. Several consultants reported similar pragmatic benefits, more than 80% of their work was repeat business. One report showed how a large company avoided a multi-million dollar purchase when the architecture of the global information system they were procuring was found to be incapable of providing the desired system attributes. In another case, involving a large engineering relational database system, a project was cancelled after 20 million dollars had been spent. The organization later learned that performance problems were largely attributable to design decisions that made integration testing impossible.

In summary, architecture evaluation enhances system documentation through explicit representation and documented design rationale. It provides a framework for understanding how well an architecture addresses critical system concerns and for early detection of software problems. While architecture evaluation does not guarantee high quality or low cost, it is an effective tool for establishing aspects of the system for which quality can be improved and budget risk reduced.

2.4 Component-Based Software Engineering Process

In order to understand architecture analysis for black-box component-based systems, it is important to first understand the software engineering processes used to develop components and component-based systems. The component-based software engineering (CBSE) process can be viewed as two separate, but related processes [Kotonya03]; development for reuse and development with reuse respectively. The first is concerned with the application domains and the development of domain-

related components. The second process is concerned with assembling software systems from prefabricated (off-the-shelf) components.

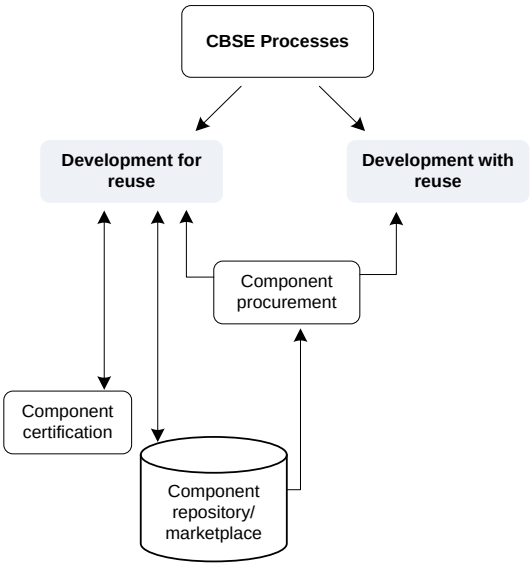


Fig. 2.3 CBSE processes [Kotonya03]

1. *Development for Reuse*: The component development process is aimed at developing generic and domain-specific components that can be made available within organization or on the open market as commercial components. To achieve successful software reuse, commonalities of related systems must be discovered and represented in a form that can be exploited in developing similar systems. Domain commonalities are used to develop models or software components that can be used to develop models system in the domain. However, developing a component is not easy as often the aim of the component developer is not to satisfy a specific requirement, but to achieve widespread reuse.
2. *Development with Reuse*: Component-based system development (CBD) is concerned with composing software systems from pre-fabricated component [Kotonya4b]. Development with reuse is mainly intended to support black-box development but should make allowances for white-box development in cases where black-box development is not feasible. It is important to recognize that for certain types of requirements and system, a black-box solution may not be

adequate or even appropriate. These two processes are related through component distribution sources and market distribution sources (see Fig. 2.3).

This thesis is mainly concerned with architecture analysis for CBSE methods that support development with reuse (i.e. for component-based system development). The next section reviews component-based system development from the point of view of three current development models.

2.4.1 COMPOSE Model

COMPOSE is an example of a process model that supports development with reuse as shown in Fig. 2.4 [Kotonya4b, Kotonya07]. The development phase implements the agenda set out in the planning phase. The first step in application development is requirements definition. Often this starts with requirements elicitation, followed by requirements ranking and modelling. This requirements process is constrained by the availability of potentially suitable components as well the nature of the application. Subsequently, the design stage partitions the service descriptions into abstract sub-systems blocks with well-defined interfaces. Sub-systems are replaced with concrete software components at the composition stage. Beyond this stage the system goes into a management cycle. Like the requirements stage, the design stage proceeds in tandem with the verification and planning phases, and may iterate to the requirements stage from time to time.

The architectural design stage partitions required functionality (i.e. services and constraints) into logical components, which can be composed using off-the-shelf components and services. The discovery and verification phase is intended to ensure that there is an acceptable match between available software components and the system being built. The negotiation and planning phase implements the necessary mechanisms for resolving conflicting system attributes and sets out the development agenda.

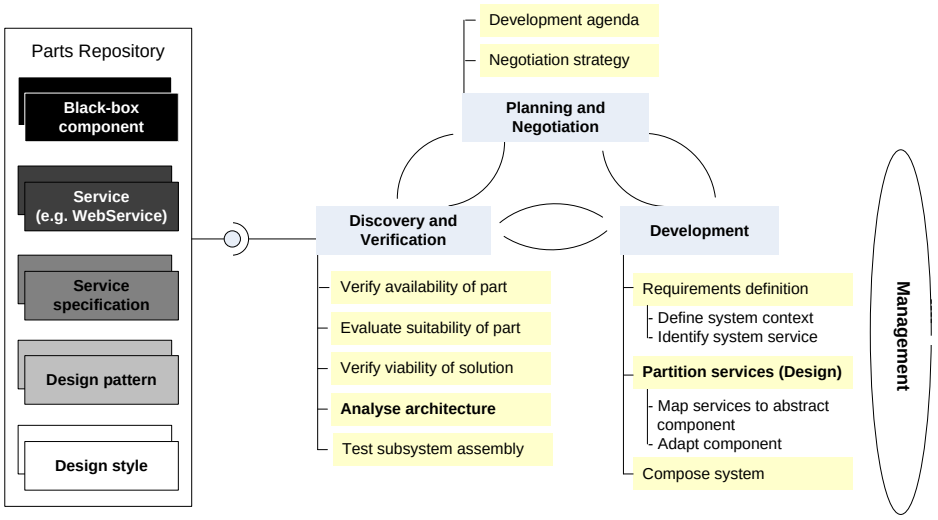


Fig. 2.4 Component-based system development [Kotonya04b]

The verification phase is intended to ensure that there is an acceptable match between selected software components and the system being built. At the requirements stage, verification is used to establish the availability of suitable software components and services, and the viability of a reuse-driven solution. At the design stage verification is concerned with ensuring that the design matches the system context (e.g. system characteristics such as requirements, cost, schedule, operating and support environments). This requires architectural analysis and black-box testing.

In summary, COMPOSE is an approach for supporting the development of black-box component-based systems from formulation through to deployment. COMPOSE is supported by a constraint-based language known as Component Architecture Description Language (CADL) [Kotonya08]. CADL provides support for partitioning services into abstract component architectures, searching and verifying plug-compatible black-box components, composing and adapting design-level components, and visualising, mediating and validating component changes. COMPOSE does not explicitly support architecture evaluation. However, it highlights the need for architecture evaluation as a pluggable process during design verification.

2.4.2 Pressman Model

According to Pressman [Pressman09] two processes occur in parallel during the CBSE process; domain engineering and component-based development (see Fig. 2.5). The intent of domain engineering is to identify, construct, catalogue, and disseminate a set of software components that have applicability to existing and future software in a particular domain. An application domain is like a product family which has similar functionality or intended functionality. The goal is to establish a mechanism by which the software engineer can share these components in order to reuse them.

Domain engineering begins by identifying the domain to be analysed. This is achieved by examining existing applications and by consulting experts on the type of application that are aiming to develop. A domain model is then realised by identifying operations and relationships that recur across the domain and are therefore candidates for reuse. This model guides the software engineer to identify and categorise components that will be subsequently implemented.

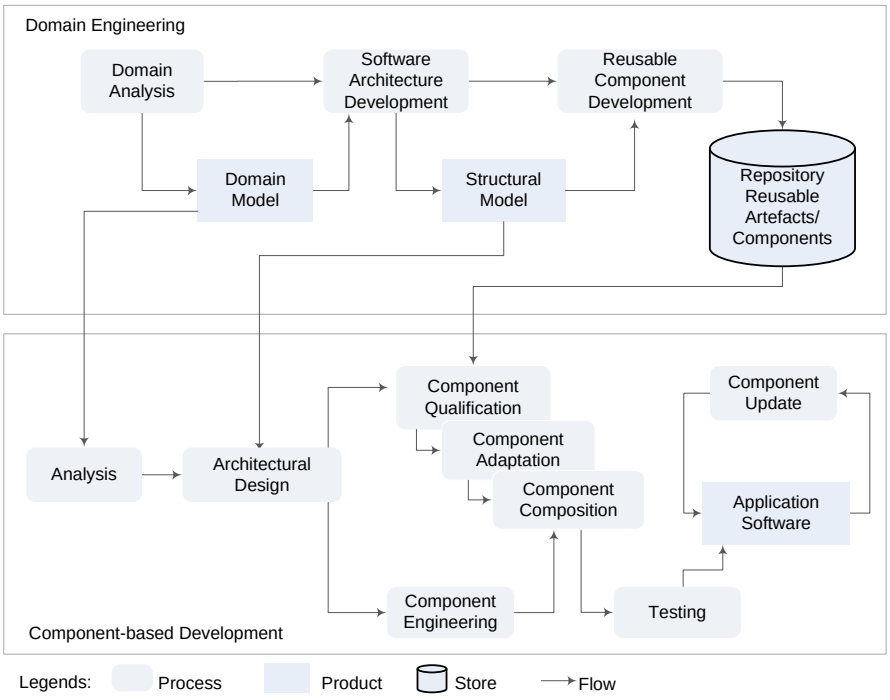


Fig. 2.5 CBSE process model [Pressman09]

Pressman describes the CBD activity as consisting of three stages; qualification, adaptation, and composition. Component qualification examines reusable components by identifying the characteristic of their interfaces. The qualification stage does not always provide the complete picture of whether a component will fit the requirements and the architectural style. However, this is a process of discovery by the software engineer to ensure a candidate component will perform the function required, and whether it is compatible with the architectural styles of the system. The three important characteristics looked at are performance, reliability and usability. The adaptation stage is required because it is rare for the components to immediately integrate with the system. Different strategies are used for adaptation such as grey-box wrapping and black-box wrapping. Grey-box wrapping relies on the availability of a component library that enables conflicts to be removed or masked. In situations where the component source-code is not available, black-box wrapping is used to adapt the component at the interface level.

The component composition stage integrates the components into a working system. This is accomplished by way of an established infrastructure to bind the components into an operational system. The infrastructure is often a library of specialised components itself. It provides a standard for the coordination of components and specific services that enable components to interact with one another and perform tasks. Common component technologies include Sun's Enterprise JavaBeans (EJB), Microsoft's .NET and CORBA's CCM [Lau07].

In summary, the Pressman model does provide any obvious means for supporting architectural evaluation. However, the model emphasis that as part of the qualification process, the software engineer should ensure that candidate components perform the required functions and are compatible with selected architectural styles. None of the component technologies mentioned provide support for architecture evaluation.

2.4.3 Brown Model

Brown [Brown96] describes CBSE as primarily an assembly and integration process that consists of five major stages; off-the-shelf components, qualification, adaptation, assembling components into system, and system evolution (see Fig. 2.6). The first stage is the process of identifying potential components that may be derived from local and remote sources. At this stage little may be known about a component's characteristic. The information available may simply its name, parameter, and required operating environment. The following component qualification stage explores detailed component documentation and specification through discovery and evaluation. Discovery identifies a component property such as its functionalities and interfaces as well as its quality aspects. The evaluation phase involves feedback gathering from other users of the components, and hands-on benchmarking and prototyping. The component adaptation stage involves wrapping three types of components; white-box, grey-box and black-box.

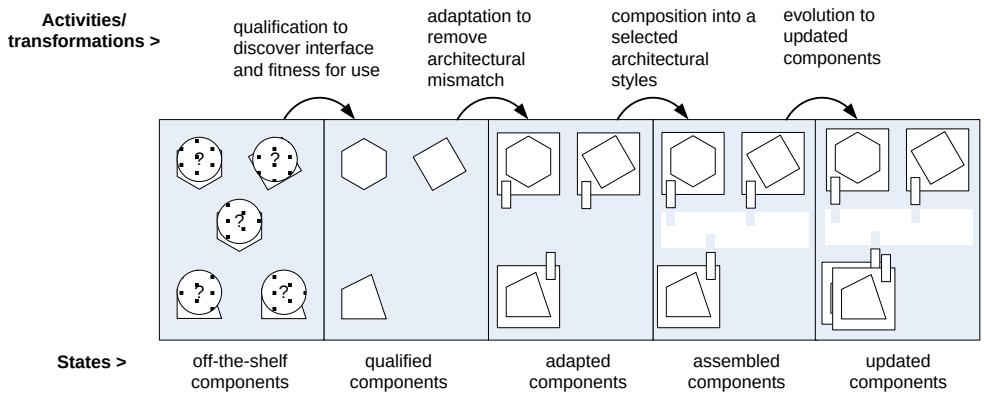


Fig. 2.6 Brown [Brown96] component-based development approach

Assembling components into system has the same objective as in the Pressman model [Pressman09], however Brown emphasizes that component composition must follow a selected architectural style. Lastly, system evolution is a process for repairing component errors, where defective components are swapped for updated ones. Similarly, when additional functionality is required, it is embodied in a new component

that is added to the system. Adding new functionality is a complex and time-consuming task. Often wrappers must be rewritten, and side effects from changes found and assessed.

In summary, the Brown model does not highlight the importance of an architectural evaluation. Nevertheless, component composition is conducted in coherence with selected architectural styles.

2.5 Summary

This chapter has provided a background to software architecture with the aim of setting the context for architecture analysis. The chapter has discussed the importance of architecture in software design and in component-based software design. The relationship between architecture and system quality has been explored and different ways of achieving quality in architecture discussed. The chapter has also explained architecture evaluation and highlighted its benefits. Lastly, three well known component-based software engineering models have been discussed and their poor support for architecture evaluation highlighted.

Chapter 3

Architectural Analysis in CBD

The importance of architectural analysis in CBD raises the need to explore how we can develop better analysis techniques and methods [Kotonya08]. The following section discusses key design challenges in CBD and identifies the necessary requirements for an architectural analysis approach in CBD [Admodisastro08]. The requirements are then used to review architectural analysis approaches that support CBD.

3.1 Design Challenges in CBD

A typical component-based system architecture comprises a set of components that have been purposefully designed and structured to ensure that they have "pluggable" interfaces and an acceptable match with a defined system context. However, the blackbox nature of many software components means there is never a clean match between system specifications and concrete software components. Services may therefore have to be re-assigned, requirements renegotiated and components adapted to achieve an acceptable match with the system context.

The design challenges in CBD can be summarized thus:

- *Balancing application context with component availability*. There is a conceptual gap between the way we articulate requirements in custom development and the reuse-driven paradigm embodied in black-box component-based system development. The features supported by commercial software solutions vary greatly in quality and complexity. This together with the variability in application contexts means that specifications delivered with black-box software components are almost always inadequate [Kotonya07]. There is need for architectural analysis approaches that facilitate the mapping of requirements to component-based system architectures by providing mechanisms that allow developers to balance aspects of requirements, characteristics of application domains, business concern and architectural considerations with the capabilities embodied in software components [Medvidovic07].
- *Pluggability*. Blackbox components are generally not tailorable or "plug and play". In addition, components may have hidden design assumptions and constraints. This has serious implications for exception handling and system quality [Kotonya05a,Crnkovic02]. The design challenge here is twofold: First, to devise ways to help the developer formulate appropriate analysis scenarios to expose structural and behavioural mismatches, and secondly, to help the developer identify and design appropriate adapters to 'repair' incompatibilities and safe-guards to minimize unforeseen side effects in the system [Crnkovic02,Stafford01].
- *Conflicting quality requirements*. Service quality constraints vary and conflict amongst themselves, and with system constraints. This makes them difficult to track and resolve. The challenge for the design process is to provide ways of assessing and addressing the adequacy of logical component configurations with respect to service and system constraints [Kotonya07,Wallnau03].
- *Evolution*. Third party software components are subject to frequent upgrades. This often leads to a disparity in customer-vendor evolution cycles and may result in unplanned upgrades being forced on the customer. In custom development,

change impact equates to the potential to make different design and coding decisions. However, for a system comprising black-box components, the decisions potentially impacted by a change are associated with the system development process, it is therefore necessary to generate information about this process as it occurs [Kotonya05b]. The design challenge here is to minimise the risks associated with change by helping system designers and integrators to understand how proposed changes may affect not only the quality of the system, but its lifecycle planning. The basis for this is effective traceability mechanisms that capture the system development history and relationships between its various artifacts.

- *Early problem detection*. A component-based system design is tightly connected to the availability of components. In addition, it is constrained by the characteristics of the application domain, business model and nature of the target platform (see Fig. 3.3). This means that design mistakes discovered late in system development may be impossible or costly to fix as decisions may already have been made on component selection. The design challenge here is to develop analysis schemes that facilitate early and incremental problem detection.

3.1.1 Necessary Requirements for Architectural Analysis

We have distilled the design challenges discussed in Section 3.1 into eight key requirements that can used to design architectural analysis methods and assess their efficacy for CBD. We outline the requirements below:

1. *Nature of Analysis*. A pluggable analysis allows the developer to adapt and tailor the design process to reflect the system context and to address domain specific needs (see Fig. 3.1) [Obbink07,Klein99].

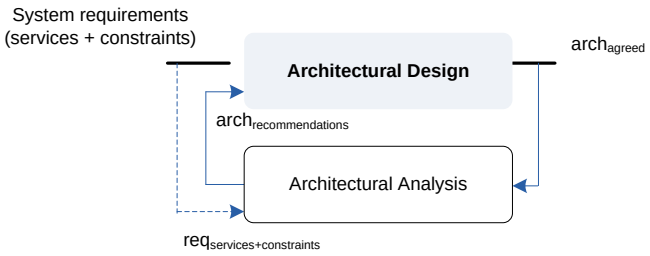


Fig. 3.1 Pluggable analysis

Fig. 3.2 illustrates the alternative embedded analysis. Because of its close binding with the design process, embedded analysis often poses problems where evaluation needs to be conducted for specific reasons such as safety analysis.

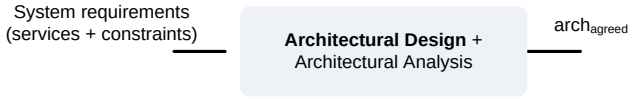


Fig. 3.2 Embedded analysis

2. *Problem Detection.* Early design problem detection cuts development costs and improves system reliability. We categorize architectural analysis schemes according to [Abowd97] as follows: early (no actual architecture exists at this stage, only preliminary design decisions), middle (architecture exists in different stages of completeness and problems associated with it can be identified) and post-deployment (both architecture and system exist, an evaluation to check whether the architecture matches the implementation can be performed).
3. *Support for Diversity.* The increasing complexity and diversity of software systems means that it is unlikely that large systems will continue to be developed using a purely component-oriented approach. Rather, a hybrid model of software development is likely to emerge where components and other solutions such as web services co-exist in the same system.
4. *Support for Negotiation.* The potential, and contextually achievable, benefits of component use must be weighed against the match between requirements and available functionality. The result is that component selection is a potentially very

complex, interdependent set of decision making problems. Support for negotiation is therefore central to successful architectural design and analysis [Hutchinson06]. As we discussed in Section 2, there is never a clean match between system requirements and concrete software components. Different design trade-offs may be required in a system architecture to achieve desired quality attributes (see Fig. 3.3).

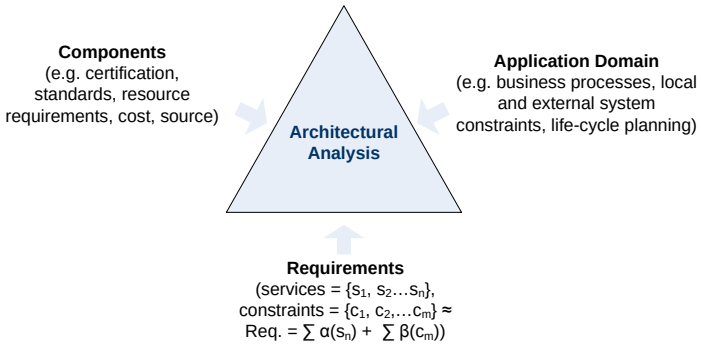


Fig. 3.3 Architectural analysis in CBD

5. *Analysis Scenarios.* Kurt Wallanau et al. [Wallnau02] describe the presence of a component in the architectural design process as a dilution of control (see Fig. 3.4). In a traditional software engineering approach a software architect makes architectural decisions based on system requirements, constraints, and business goals alone. After the system architecture plans are stabilized a set of components are evaluated. This sort of approach is not suitable for component-based systems. There may be no suitable components available to suit the specific needs of the envisioned system. By choosing to use components an architect takes on additional risk that he or she cannot control. In essence the component adds a new source of control, thus diluting the control relationship between the stakeholders' needs and the system's requirements. The changes that might occur to a component are more than that just its features and functional capabilities. Component vendors make frequent decisions about which features remain and which are removed from future release.

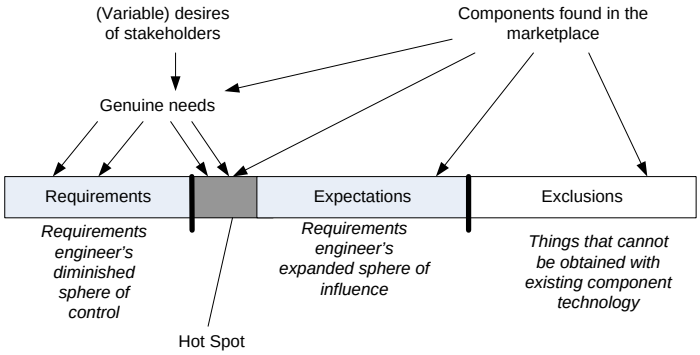


Fig. 3.4 Effect of components on spheres of control [Wallnau02]

Analysis scenarios are essential in helping the system designer understand how proposed architectural configurations and system changes might affect not only the quality and operation of system, but also its life-cycle planning [Babar04a,Ekstedt02,Weiss01]. In summary, an architectural analysis method should provide:

- Guidance for formulating and constructing analysis scenarios.
- Support for standard/portable descriptions of the system architecture (e.g. UML and XML). Rami et al. [Rami03] have highlighted ADLs as a potential instrument for supporting software architecture evaluation.
- Support for augmenting architectural descriptions with specific constraints and other information to tailor the analysis to specific questions (e.g. quality attributes, application domain characteristics and business concerns).
- Support for formulating "what if" analysis (static and dynamic) under conditions of uncertainty that allow developers to describe scenarios to assess the impact of competing designs.
- Support for evolution through qualitative and quantitative analysis that allow designers and maintainers to develop change scenarios to assess the impact of proposed changes.
- Support for stakeholder involvement in architectural analysis can help identify and resolve conflicts, assess alternatives and build consensus on priority issues. Fig. 3.5 shows the typical stakeholder roles in CBSE. Stakeholder may also

include other decision makers within and outside the organisation (e.g. regulatory bodies).

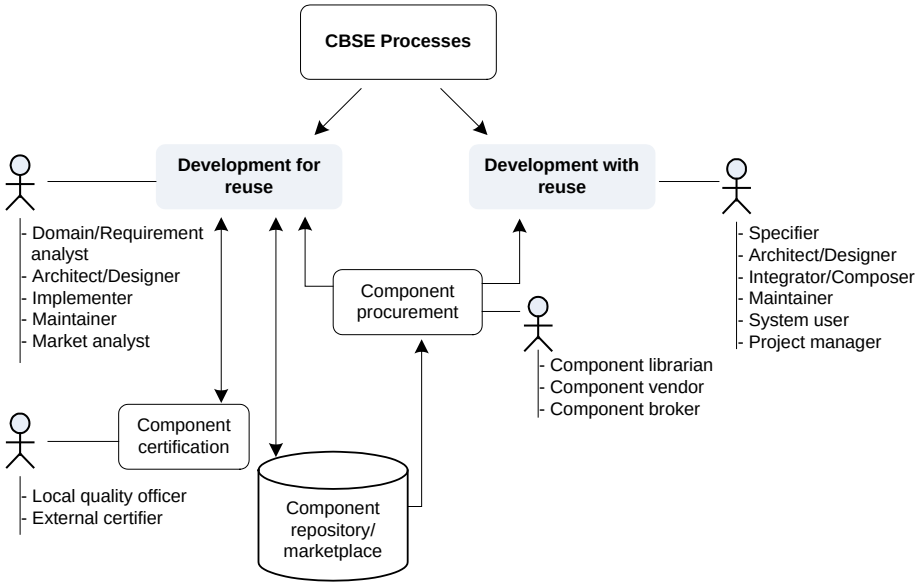


Fig. 3.5 Component and application development processes – together with associated stakeholder roles

6. *Assessment.* Architectural assessment allows the developer to establish how well a proposed system design satisfies its application and business contexts. The result of the assessment process contributes towards regression testing, impact analysis and traceability activities that may be conducted later in the development process. There are several architecture assessment techniques including use-case scenarios, conformance to patterns, metrics and organization-specific assessment techniques. Use case scenarios provide information on system contexts and logical connections [Jacobson97]. Design patterns and styles can be used to check if architectures and configurations conform to certain structural and behavioural characteristics [Babar07]. Metrics provide useful quantitative information related to interface complexity, size, component dependency and other measurable system attributes. In summary, an ideal assessment technique should reveal:

   - Structural mismatches. Incompatibilities in the data exchanged between components and verify architectural adherence to design heuristics and rules.

- Quality mismatches. Inconsistencies and mismatches between quality attributes and services and the system context. When we understand desired service and system qualities before a system is built, the likelihood of selecting or creating the right architecture is improved.
- Behaviour mismatches. Semantic mismatches between provided and required interfaces and defects in dynamic component interaction.

It is important that assessment techniques support both qualitative and quantitative analysis. Qualitative measurements provide a means for representing quality concerns in a subjective evaluation which allows logical reasoning, whilst quantitative analysis provides a mechanism to elicit subjective responses from the stakeholders that provide empirical and measurable values.

7. *Maturity*. Maturity indicates the state of readiness of architectural analysis approaches to be adopted in an organization. An important metric for measuring maturity is validation results [Dobrica02,Babar04b]. We use a CMM-like [Persse01] approach to categorize the maturity as follows: initial (approach has not being validated), repeated (validation through limited complexity and domains with consistent published results) and defined (validation through various complexity and domains with consistent published results).
8. *Tool support*. Architectural analysis is a complex activity that involves the planning, analysis, negotiation and assessment of large amounts of interrelated, often conflicting information. A tool should provide support for extracting architectural definition, storing architectural knowledge, analyzing architectural design decisions, identifying trade-offs and offering alternatives [Babar04b,Obbink07,Kazman96, Bashroush04].

In the next section we use these requirements to assess architectural analysis approaches intended to support component-based development.

3.2 Architectural Analysis Approaches

3.2.1 NFR-Framework

Chung [Chung95a] proposes a process-embedded framework for generating architectural fragments by evaluating non-functional requirements against stored design knowledge. The approach is associated with a prototype tool called NFR-Assistant [Tran99]. Fig. 3.6 shows the NFR-Framework process. In the approach, non-functional requirements are represented as goals to be addressed and achieved during the process of architectural design. Each goal is associated with a "type", a parameter list and importance (e.g. Modifiability [system: critical]).

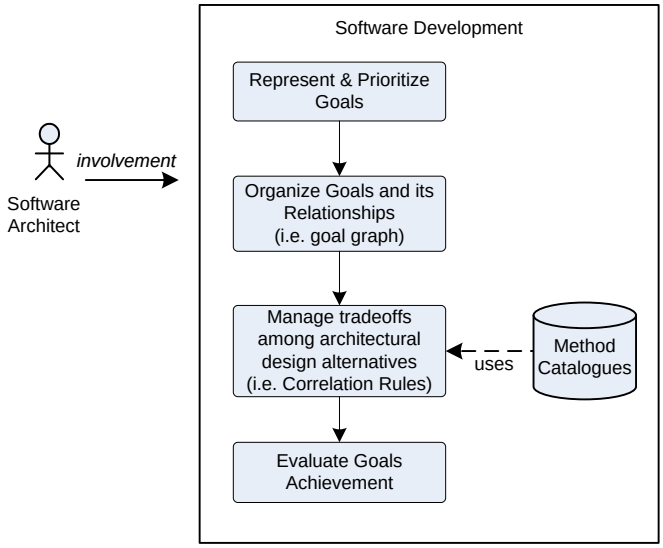


Fig. 3.6 NFR-Framework people and activities

NFR goals have the property of potentially interacting with each other, in conflict or in synergy. This property is used to systematically guide selection among architectural design alternatives and to rationalize the overall architectural design process. Goals (nodes) and goal relationships (links) also correspond to design alternatives, decisions, and rationale. They are recorded and structured in a goal graph with link types annotated as either "AND" and "OR".

Architectural design knowledge and experience about specific NFRs is organized into methods and made available to the software architect through systematic search. These methods are categorized into three types as the follows:

1. Decomposition methods are used for refine or clarify NFRs. For example performance can be decomposed into space and time.
2. Satisfying methods are used to organize knowledge about achieving NFRs goals where they are embedded in the architectural design. For example, an implicit function invocation style can be used to hide implementation details in order to make an architectural design more extensible, thus contributing to goals that required these NFRs.
3. Argumentation methods are used to organize principles and guidelines for making design rationale for or against a design decision. Argumentation methods act as determiners to verify which goals are most important to satisfy, and in selecting among alternatives to satisfy NFR goals, especially in the context of time and effort constraints.

Correlation rules that embed knowledge and experience about design trade-offs are used by the software architect to select among architectural alternatives. For example, correlation rules showing the contribution of architectural design alternatives for (+) or against (-) specific NFRs. An entry with +– denotes an uncertain contribution, and requires the software designer to consider the characteristics of the intended application domain. Throughout the goal expansion process, the evaluation procedure propagates upwards, via the label of nodes in the graph. The effect of each design decision from child to parent nodes provides an assessment of the degree of goal achievement. An assessment is carried out by relating this to the characteristics of the intended application domain. NFR-Framework has been used and validated in Information System domain namely Credit Card System, Health Insurance System and Government Cabinet and Tax Appeals system [Chung95b].

3.2.2 REDEPEND-REACT

REDEPEND-REACT is an architectural analysis tool that supports the i* approach which is represented in Strategic Dependency models (SD) [Grau05,RedependReact07]. i* is an actor modeling language that is used to represent software domains and actors (human, organization, hardware or other software). SD describes a network of dependency relationships amongst various actors in an organization context. Actors are represented by nodes; links between nodes represent dependencies between actors. The depending actor is called Depender and the actor who is depended upon is called the Dependee. The approach is shown in Fig. 3.7.

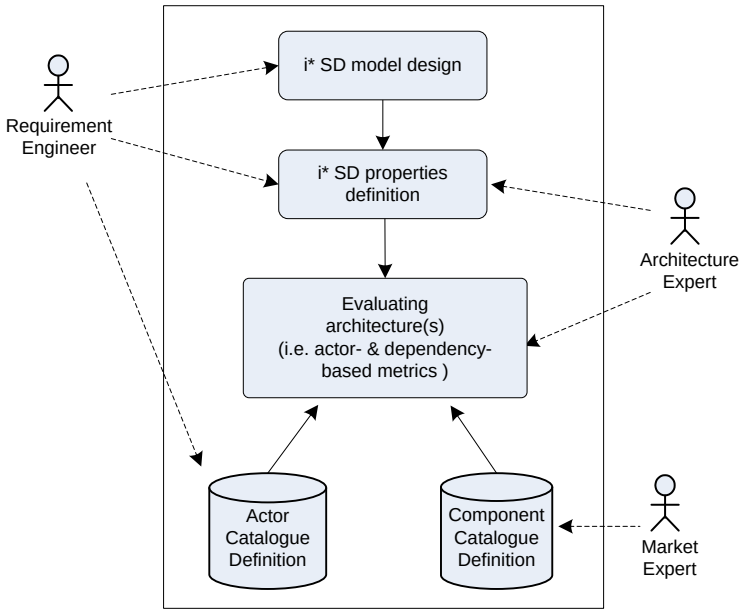


Fig. 3.7 REDEPEND-REACT architectural analysis process

REDEPEND-REACT provides guidelines for formulating metrics over i* models that a developer can use to perform architectural analysis. The metrics are selected with respect to properties that are important to the system being modeled (e.g. security, efficiency or accuracy). Metrics are defined in terms of the actors and dependencies in the models, and the results of the evaluation are used to inform multiple component

selection. Metric measurement is performed using a MS Excel[1™] tool which allows the user to define additional metrics and to modify actor values interactively. As the values on the architectures are formulas based on these values, the results are automatically updated. REDEPEND-REACT has been successfully used to analyse several information management system case studies including; a Meeting Scheduler system, an e-Learning system and an e-Business system.

3.2.3 ATAM

The Architecture Trade-off Analysis Method (ATAM) [Kazman98,Kurpjuweit02] is a pluggable scenario-based approach. ATAM focuses on multiple quality attributes (currently; modifiability, availability, security, and performance). It is aimed at locating and analyzing trade-off points for areas of highest risk in the architecture. Attribute-specific questions generated using scenarios of interest are used to identify possible architectural solutions to achieve desired system quality attributes. The analysis process derives three architectural decisions (i.e. sensitivity points, trade-off points and risks) that have marked effect on one or more quality attributes. ATAM requires the participation and mutual cooperation of three groups of stakeholders: an evaluation team that is external to the project, project decision makers, and architecture stakeholders.

The approach requires the architect to walk through each high-priority attribute-specific scenario, showing how it affects the architecture (e.g. modifiability) and how the architecture responds to it (e.g. for quality attributes such as performance, security and availability). If the system has complex quality attribute requirements or is in a complex and unusual domain, specialists may be needed to augment the expertise of the core evaluation team. Along the way, the evaluation team documents the relevant

[1™] MS Excel is a trademark of the Microsoft Corporation

architectural decisions, and identifies and catalogues their risk, non-risks, sensitivity points and trade-off.

Sensitivity points are parameters in the architecture to which some measurable quality attribute is highly correlated. To find the trade-off, all important architectural elements with multiple sensitivities are located. For example the number of copies of a database might be a sensitivity point for both availability and performance. Fig. 3.8 shows how the ATAM activities are partitioned into four iterative phases. ATAM has been extensively evaluated in different application domains including embedded [Kazman98] and general information systems [Bass05].

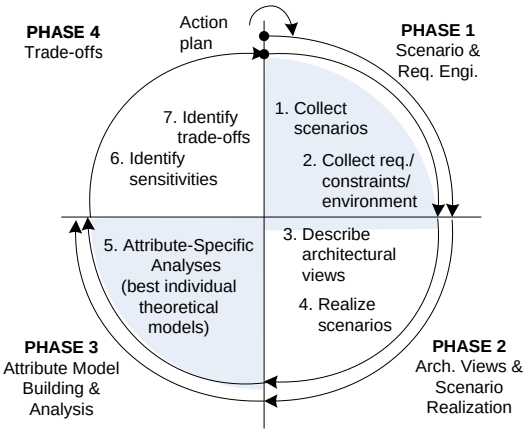


Fig. 3.8 ATAM activities [Kazman98]

3.2.4 ASAAM

Aspectual Software Architecture Analysis Method (ASAAM) is scenario-based architecture analysis method that is able to identify concerns that can be easily localized and specified in architectural abstraction, and identify concerns that crosscut various architectural components [Tekinerdogan04]. For example, failure management aspects, monitoring Aspects and operating system aspects are inherently crosscutting concerns. The method is associated with a prototype tool called ASAAM-T. Architectural analysis activities for ASAAM are shown in Fig. 3.9.

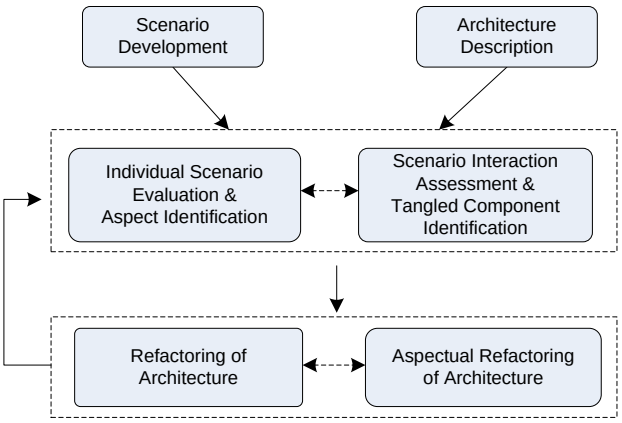


Fig. 3.9 ASAAM process [Tekinerdogan04]

ASAAM takes as input a problem description, requirements statement and architecture descriptions. In scenario development stage, scenarios from various stakeholders are collected, which represent both important uses and anticipated uses of the software architecture. A scenario is considered as a brief description of some anticipated or desired use of the system. ASAAM starts characterizing scenarios that can be directly supported by the architecture (direct scenarios) and scenarios that require the redesign of the architecture (indirect scenarios). Some scenarios, however, can be scattered over different architectural components and their impacts are difficult to localize in individual components.

ASAAM introduces a set of heuristic rules to identify these so-called aspectual scenarios, and to derive architectural aspects based on domain model developed through a domain analysis process. Based on detailed impact analysis for a given set of scenarios, ASAAM provides a categorization of the architectural components into cohesive components, composite components, and tentative tangled components. Tentative tangled components are component that perform semantically distinct scenarios and cannot be decomposed. The results of the detailed impact analysis can be used in aspect-oriented design and aspect-oriented programming. ASAAM is at the initial stage of maturity with no significant case studies.

3.2.5 Chaining Framework

Stafford et al. [Stafford01b] propose a static dependency analysis approach at architectural level called chaining. The approach uses the Rapide ADL specification [Luckham95]. Dependence analysis is widely used at implementation level to aid program optimization (i.e. anomaly checking, program understanding, testing and debugging). The chaining framework uses this technique to analyze architectural designs by taking a broader view of dependence relationships that are more appropriate to the concerns of architectures and their component interaction.

Dependence at the architectural level arises from the interconnections among components and the constraints on their interaction. These relationships may involve some form of control or data flow, but more generally they involve source structure and behaviour. Source structure is related to the static source specification dependencies, while behaviour is related to dynamic interaction dependencies.

The chaining framework provides analysis of structural and behavioural aspects of system architecture using a tool called Aladdin [Stafford98]. The framework describes three types of chaining (see Fig. 3.10):

1. Affected-by chains: Consists of the set of components and/or their elements that could potentially affect an element of a component, C. These are elements that C is affected by.
2. Affects chains: consist of the set of components and/or their elements that could be affected by a component, C. These are elements that C affects.
3. Related-to chains: consists of the set of components and/or their elements that may affect or be affected by an element of a component, C. This chain is the combination of the affected-by and the affects chains for elements of Component, C.

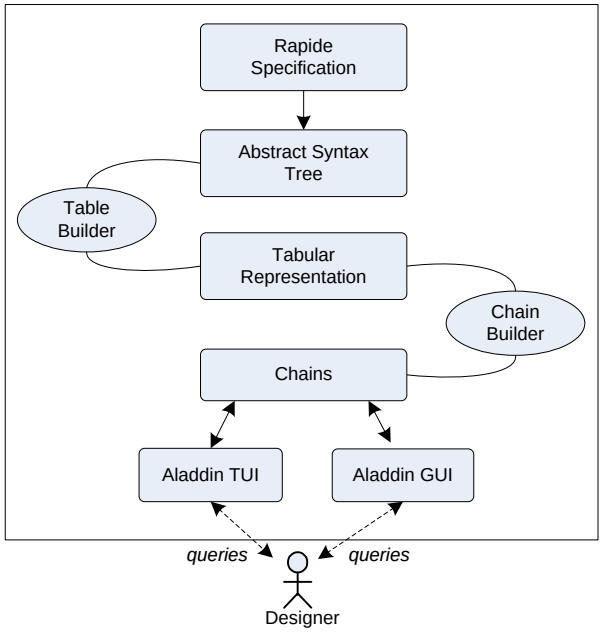


Fig. 3.10 Chaining Framework

Aladdin generates a dependency table that is built from an abstract syntax tree that represents the set of relationships that exist between pairs of elements in the architecture. Aladdin also provides a set of queries over the chains (through both a graphical and a textual user interface) that aid in answering dependency questions. By performing analysis using these queries, anomalies can be revealed. However, only the experience of software engineer can determine whether the anomalies are actual faults in the specification. For instance, it is possible that an unused event has been included in an interface because it is expected to be needed in the future, not because it is a misconnection. The Chaining Framework at the initial stage of maturity which evaluated using a small case study of a gas station system.

3.2.6 ARGUS-I

ARGUS-I [Vieira00] is a specification-based analysis tool which uses the C2-style architecture description language [Medvidovic96] and augments it with component behaviour specification using Statecharts. The ARGUS-I tool performs analysis at

component and architectural level. Component-level specification analysis allows for static (i.e. interface inconsistencies and component-Statechart inconsistencies) and dynamic analysis (i.e. enables the execution of component Statecharts). The analysis process is shown in Fig. 3.11.

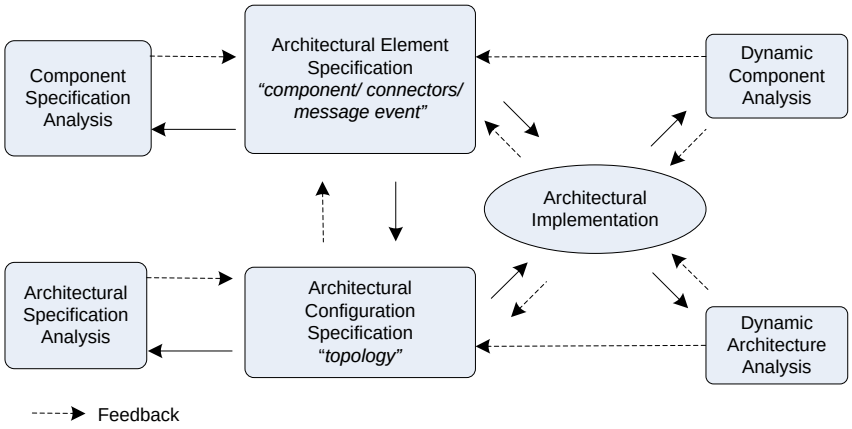


Fig. 3.11 ARGUS-I process [Vieira00]

Architecture-level specification checks are performed statically by verifying structural and behavioural dependencies among components, and dynamically by evaluating architecture configuration through simulation. The analysis capabilities of Argus-I have been illustrated using a medium-sized Elevator Control System example.

3.2.7 Odyssey-Adapt

Odyssey-Adapt is a plug-in for the Odyssey IDE [Spagnoli06] that supports CBD in both domain engineering and application processes. Most of the analysis is focused on the component interface that is intended to support component adaptation and composition during development. The approach uses three design patterns (proxy, façade and adapter) to tackle component interface mismatches and structural complexity.

Fig. 3.12 shows the analysis process. The approach defines two types of dependencies between a provided and a required interface; assembly connector and

incompatibility dependency. An assembly connector dependency represents the actual composition between two components through their interfaces. An incompatibility dependency shows the relationship between two components that require some kind of adaptation before their interface can be composed.

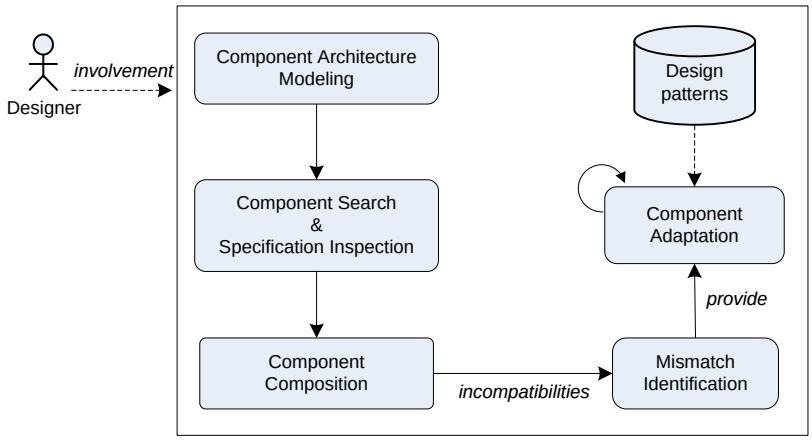


Fig. 3.12 Odyssey-Adapt

Whenever a provided and a required interface are related, Odyssey-Adapt triggers the incompatibility detection function. Three types of incompatibilities are considered:

1. Structural. These are conflicts related to syntactic problems between a provided and required interface. These include interfaces with different names, interfaces with methods that differ in their signature, interfaces with different numbers of method, and any combination of these three. They are automatically discovered by a detection function that compares the specification of the interfaces.
2. Behavioural. These are semantic mismatches between the provided and required interface. This mismatch identification process is the responsibility of the designer, which means that all conflicts are documented manually in an incompatibilities note and tagged with the provided interface.

3. Hybrid. These are mismatches that occur from combination of structural and behavioural incompatibility. This type of mismatch is automatically detected, provided that the behavioural incompatibility has been previously marked.

Odyssey-Adapt is a relatively new approach and has not been validated on a significant software system.

3.2.8 Engineering Framework

Becker et al. [Becker06] have proposed an adaptation process for detecting and resolving component mismatches based on a taxonomy of design patterns. The adaptation process is applied during architectural design, whenever an analysis of the system indicates a mismatch between two constituent components. The taxonomy contains five distinct classes of component mismatches; technical, signature, protocol, concept and quality. These are associated with patterns that may overcome the mismatches.

The adaptation process has five steps as follows (see Fig. 3.13):

1. *Detect mismatches*. Find the mismatch between the required and provided interface.
2. *Select measure to overcome the mismatch*. Select from the established patterns the one which is known to solve the specific mismatch.
3. *Configure the measure*. Often the pattern selected is fine-tuned as patterns are described as abstract solutions to the problem. Therefore, utilize relevant specification and query developer for additional input.
4. *Predict the impact*. Predict the impact of the solution on the existing setting.
5. *Implement and test the solution*. If the prediction indicates that the mismatch is fixed, the solution is implemented, either by systematic construction or by using generative technologies.

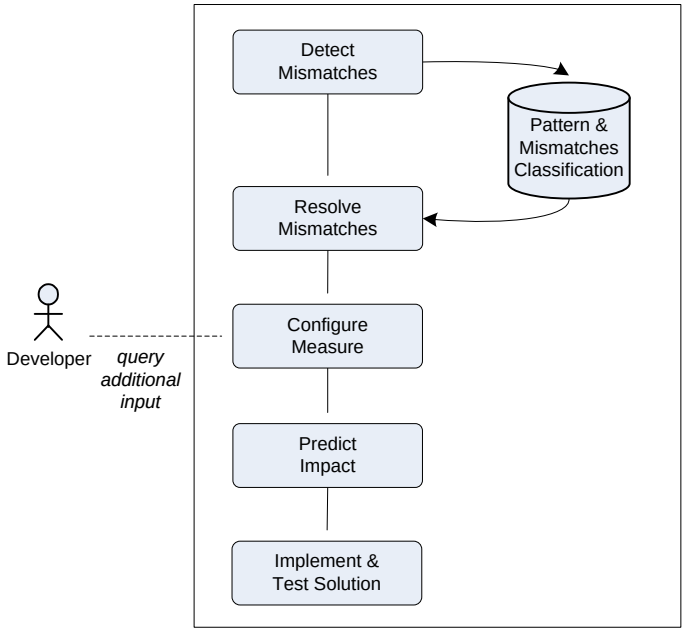


Fig. 3.13 The process of adapting a component

The Engineering Framework has been partially evaluated using a small case study of a water cooling system.

3.3 Methods Summary

The results of the assessment are summarised in Table 3.1. Briefly, the NFR-Framework is an embedded, early problem detection approach that supports whitebox development. It supports a negotiation process that is concerned solely with trading-off non-functional attributes. Central to the negotiation process is the system architect. The NFR-framework provides some limited help with formulating analysis scenarios and allows the developer to uses quality attributes to explore and verify design goals. It is tool supported and supports both qualitative and quantitative assessment.

Table 3.1 Comparison of architectural analysis approaches

REQUIREMENT \ ANALYSIS APPROACH			NFR-Framework	REDEPEND-REACT	ATAM	ASAAM	Chaining Framework	Argus-I	Odyssey-Adapt	Engineering Framework
Nature of Analysis	Pluggable		○	○	●	●	●	●	○	●
	Embedded		●	●	○	○	○	○	●	○
Problem Detection	Early		●	●	○	○	○	○	○	○
	Middle		○	○	●	●	●	●	●	●
	Post-deployment		○	○	○	○	○	○	○	○
Diversity	Component[1]		○	●	◐	○	○	○	○	●
	Hybrid		○	○	○	○	○	○	○	○
Negotiation Support (Trade-off analysis)	Help with formulation		◐	◐	◐	◐	○	○	○	○
	Quality attributes		●	●	●	●	●	○	○	●
	Business concerns		◐	○	●	○	○	○	○	○
	Application domain characteristics		●	●	●	◐	○	○	●	○
	Component features		○	●	◐	◐	●	●	●	◐
Analysis Scenario	Help with formulation		◐	◐	◐	◐	○	○	○	○
	Support for augmentation		○	○	◐	○	○	◐	●	●
	Stakeholder involvement	Project Manager	○	○	●	○	○	○	○	○
		Architect/ Designer	●	●	●	◐	●	●	●	◐
		Evaluator	○	○	●	○	○	○	○	○
		Component Provider	○	●	○	○	○	○	○	○
	'What-if' analysis		●	●	●	●	○	○	○	○
Assessment	Structural		◐ (Ql, Qt)	● (Qt)	● (Ql, Qt)	● (Ql)	● (Ql)	● (Qt, Ql.)	● (Ql)	◐ (Ql)
	Behavioural		○	○	● (Ql, Qt)	● (Ql)	● (Ql)	● (Qt, Ql.)	◐ (Ql)	◐ (Ql)
	Quality attributes		● (Ql, Qt)	● (Qt)	● (Ql, Qt)	● (Ql)	○	○	○	○
Maturity[2]			◐	◐	◐	○	○	○	○	○
Tool support			●	●	◐	●	●	●	●	◐

● Supported/ [1]Blackbox support/ [2]Defined ◐ Partially Supported/ [1]Greybox support / [2]Repetition ○ Not Supported/ [1]Whitebox support / [2]Initial

Qt. – Quantitative assessment Ql. – Qualitative assessments

The REDEPEND-REACT approach is a maturing, embedded approach that supports blackbox development. The approach intended for early problem detection and provides good support for negotiation. It also provides extensive help with formulating analysis scenarios and involves three different system stakeholders in the analysis. It is tool supported and provides good quantitative assessment for structural and quality attributes analysis. It is significantly weak in behavioural analysis.

ATAM is a maturing approach that is pluggable, supports greybox development and has extensive support for trade-off analysis (i.e. quality attributes and business concerns). ATAM focuses on middle problem detection and provides good help with formulating analysis scenarios. However, it provides only partial support for augmenting of architectural descriptions and experimentation. It is tool supported, and provides both qualitative and quantitative assessment for structural, behavioural and quality attributes analysis.

The ASAAM is a pluggable, scenario-based method that supports whitebox development. Like ATAM, it is a middle analysis method. It has relatively good support for trade-off analysis (quality attributes and components), but poor support for stakeholder involvement. It provides limited support for formulating analysis scenarios, but good support for "what-if" analysis. It is tool supported provides qualitative assessment for structural, behavioural and quality attributes analysis.

The chaining approach is a pluggable architectural analysis approach that supports whitebox development. The approach is intended for middle problem detection. It provides limited help with formulating analysis scenarios and relies on the experience of the software engineer to verify behavioural anomalies. It is tool supported and provides qualitative assessment for structural and behavioural analysis.

ARGUS-I is a relatively new, pluggable, middle approach that supports whitebox development. ARGUS is tool supported and provides good qualitative and

quantitative assessment for structural and behavioural analysis. However, it provides limited help with formulating analysis scenarios and has poor support for negotiation.

Odyssey-Adapt is a relatively new, embedded architectural analysis process for the Odyssey development environment. It supports whitebox development and is intended for middle problem detection. The analysis is largely structural and limited to component interface mismatches. There is no provision in the method for analysing non-functional properties and no support for negotiation. Limited support is provided in method for formulating analysis scenarios. The resulting assessment is a qualitative report detailing structural, behavioural and hybrid mismatches. However, the behavioural mismatches are weakly identified and tackled.

The Engineering framework is an immature, pluggable, middle analysis method that supports blackbox development. Its support for negotiation is limited to quality attributes. The framework provides limited support for both structural and behavioural aspects of design. The resulting assessment is qualitative. In our view, the Engineering framework is still at an early stage of development. Its guidelines for component adaptation are very generic and it relies heavily on designer experience to achieve there's considerable reliance on designer experience as the steps above indicate.

3.4 Summary

Many of the challenges in component-based development arise because components already exist before the system is developed. The need to trade-off and accept compromise is therefore central to the successful development of component-based systems. However, current architecture analysis approaches provide poor support for negotiation. The chapter also highlighted the poor support for diversity in current architecture analysis approaches. Current approaches are largely designed to support a particular type of analysis (e.g. structural or conformance checking) and often for a specific application domain. However, the black-box nature of the software

components, and the variability in stakeholder concerns and application contexts, means that there is value in diversity in analysis. Critically, none of the approaches reviewed in this thesis support hybrid reuse-driven development, even though, increasingly applications are being developed for which different types of reusable software co-exist in the same system (e.g. OTS components and services).

Support for stakeholder involvement in architecture analysis can help identify critical system concerns and conflicts, assess alternatives and build consensus on priority issues. In current architecture analysis approaches, the role of architectural design is left largely to the system designer. However, system stakeholders often include decision makers within and outside the organisation (e.g. regulatory bodies). Effective analysis must be able to identify, express and analyse concerns from different system stakeholders.

Most of the existing architecture analysis techniques are based on proprietary notations and provide limited support for converting architectures described standard modelling notations such as UML. This means that many architectural designs have to be described anew in the proprietary notation. Lastly, current architecture analysis approaches are difficult to incorporate into existing design processes without significant disruption or changes to the existing processes. It is important that an architecture analysis approach causes as little disruption as possible to the existing process.

The chapter discussed architecture analysis problems in component-based development and identified the necessary requirements for architectural analysis approaches. The requirements have been used to assess eight existing architectural analysis approaches intended to support component-based development. The results of the assessment are summarised in Table 3.1 and published in [Admodisastro08].

Chapter 4

Component-based Software Architecture Analysis Framework

In Chapter 3, I highlighted the poor support for component-based system design issues in current architecture analysis approaches. I noted that most architecture analysis approaches are designed to support custom rather than black-box software development, making them inappropriate for addressing the unique design problems posed by black-box development [Kotonya08]. This Chapter describes my proposed solution, Component-based Software Architecture analysis FramEwork (CSAFE), which is intended to address the problems discussed in Chapter 3.

4.1 The Framework

CSAFE is a scenario-driven, negotiation-based architecture analysis approach intended to support black-box development. However, while CSAFE is primarily intended to support black-box development, we recognise that there might be aspects of the system for which a black-box solution is not feasible or appropriate. CSAFE supports custom

development in such situations by treating abstract design components as placeholders for custom development.

An iterative analysis process and an integral toolset underpin CSAFE. The analysis process is supported by an architecture description language, iXML ADL, and extensible repository of architecture design templates and component specifications. The iXML ADL defines the architectural elements, their relationships and the rules that govern valid architectural descriptions. The architecture design templates specify configurations that embody specific design goals and best practice, while the component specifications represent salient properties of concrete components. Lastly, CSAFE is process-pluggable rather than embedded to minimise disruption to the development process. Fig. 4.1 shows how CSAFE plugs into a typical development process.

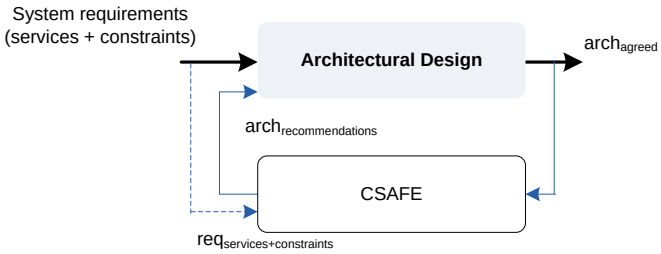


Fig. 4.1 CSAFE and architectural design process

The CSAFE approach comprises 4 iterative steps as shown Fig. 4.2:

1. Identify system or sub-system architecture to analyse.
2. Formulate analysis scenario(s) by identifying and prioritising quality concerns as goals to be addressed and achieved during architecture analysis.
3. Analyse architecture based on analysis scenario and available components.
4. Modify architecture according to recommendations
5. Repeat step (1) until done

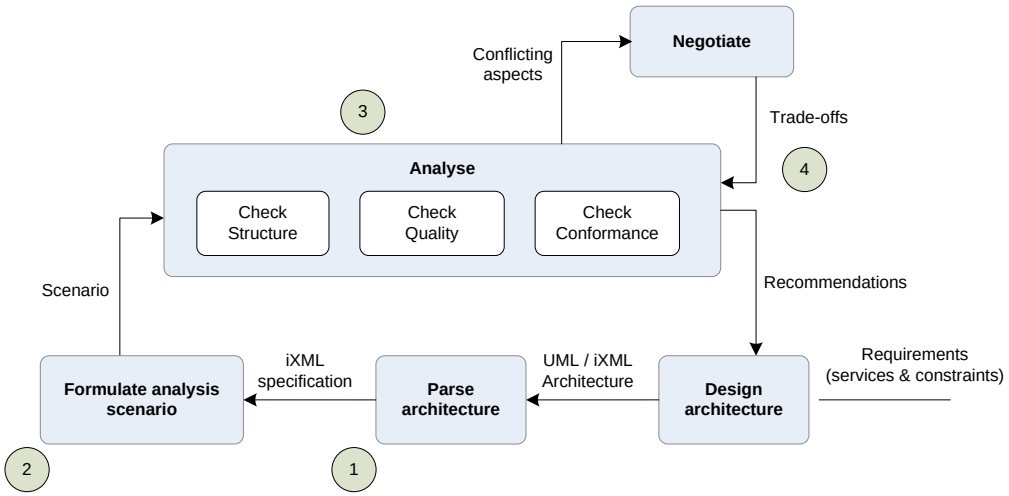


Fig. 4.2 Architecture analysis process

The architecture design stage is concerned with the construction of the system architecture. The CSAFE analysis process accepts architectures expressed in the standard UML component notation [Uml10] or in the iXML architecture description language. iXML is an XML-based ADL developed to support analysis in CSAFE. The iXML ADL is discussed in detail in section 4.1.2.3. Architectures expressed in UML are converted into iXML specification to allow for machine processing. The iXML ADL serves three purposes; first, it allows both pre-existing and new architectures to be analysed. Secondly, it allows for a portable, platform independent description of the system architecture. Lastly, it provides the system designer and other stakeholders with a mechanism for augmenting architectural descriptions to explore "what if" analysis.

Scenario formulation is essential in helping the system designer verify how closely a proposed architectural solution matches desired system attributes, and to understand how system changes might affect not only the quality and operation of the system, but also its life-cycle planning. Analysis scenarios provide a means for augmenting architectural descriptions with specific constraints and other information to tailor the analysis to explore specific questions (e.g. quality attributes, application domain characteristics and business concerns). Analysis scenarios also allow designers to

formulate "what if" analysis under conditions of uncertainty to assess competing designs and change impact.

The analysis process (step 4, in Fig. 4.2) allows the developer to establish how well a particular system design satisfies its application and business contexts. The analysis process uses standard and user-defined architecture design templates, component specifications and a process of negotiation to identify an architectural configuration that offers the best balance between critical stakeholder concerns and available component functionality. The output of the analysis process is a report outlining potential inconsistencies and mismatches, and recommendations for improving the architecture. The next sections discuss each of the stages of the CSAFE process.

4.1.1 Weaving Requirements and Architectural Design

In addition to analysing pre-existing architectures, CSAFE also allows the system designer to derive architectures from scratch using a service-oriented requirements method based on the notion of viewpoints that maps requirements onto the iXML ADL. The requirements method has been adopted from [Kotonya04b] and adapted to work with CSAFE [Admodisastro11a]. A viewpoint is a perspective of the software architecture from a requirements or analysis standpoint (see Fig. 4.3).

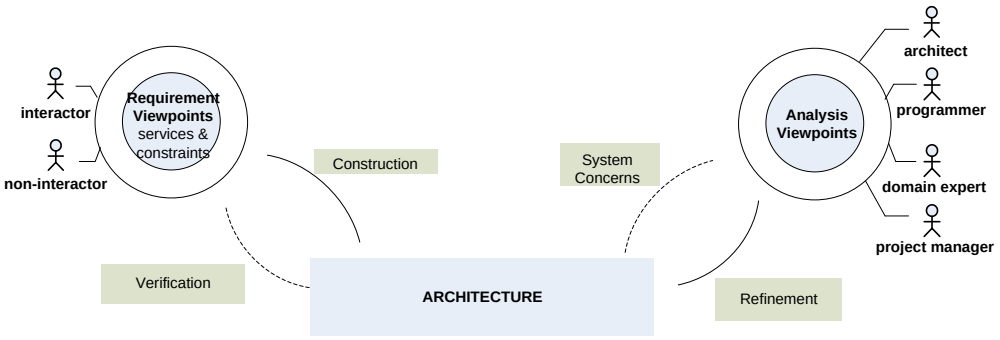


Fig. 4.3 Requirement and analysis viewpoints

Requirements viewpoints identify requirements sources and analysis viewpoints identify the human actors involved in the analysis of the architecture as follows:

- *Requirement viewpoints* represent sources of requirements. They are grouped into Interactor and Non-interactor viewpoints. Interactor viewpoints comprise operator and component viewpoints. Operator viewpoints map onto classes of users who interact with the proposed system. Component viewpoints correspond to software components and hardware devices that interface with the proposed system. Non-interactor viewpoints are entities that do not interact directly with the intended system, but which may express an interest in the system requirements. Non-interactor viewpoints provide a mechanism for expressing critical 'holistic' requirements, which apply to the system as a whole. Fig 4.4 shows the typical requirements types associated with the different classes viewpoint.

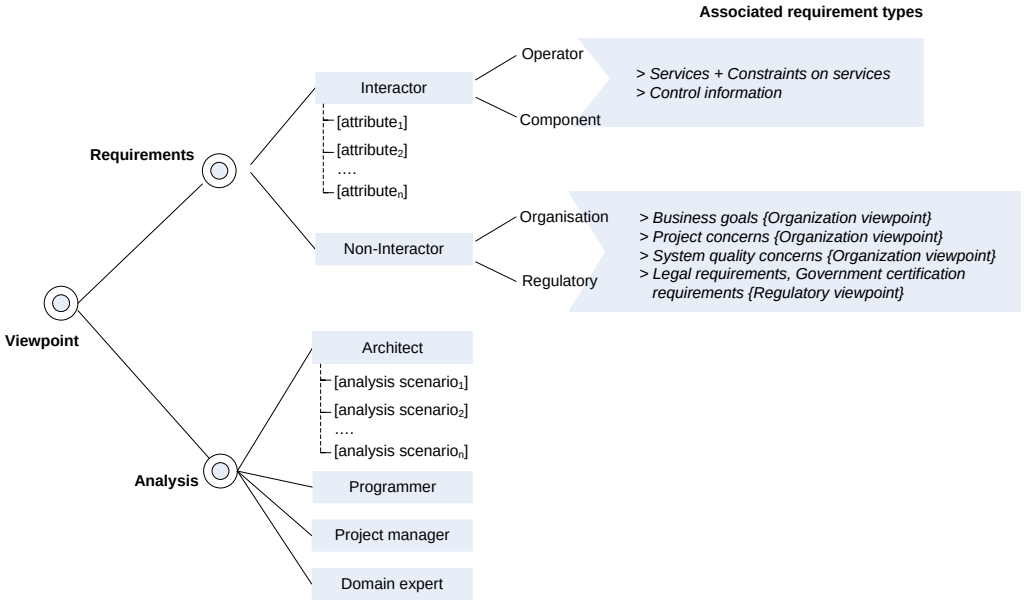


Fig. 4.4 Abstract viewpoint structure

In this thesis a requirement is defined as a statement of system service or constraint [Sommerville10]. Services represent expressions of functionality, both required and offered and, crucially, expressed in a way that shows how available

components satisfy what is required [Kotonya05a]. Constraints represent stakeholder concerns such as component cost, certification, memory and platform restrictions, or dependability requirements such as security, performance and availability. They may also represent elements of interdependence that are introduced to allow services to meet certain architectural considerations (e.g. Service X and Service Y may not reside in the same component). Finally, constraints may capture dependencies that are introduced to make certain component choices acceptable in the current context, particularly with regard to the outcome of negotiation and thus may hold important design rationale information.

Modelling services from the point of view of viewpoints also exposes interesting interrelationships between services and constraints and raises questions about how best to address this in the architecture. In Fig. 4.5, for example, two viewpoints express different availability constraints on the same service. Service 3 in actor viewpoint 1.1 has an availability requirements of 98% or greater, while actor viewpoint 1.2 has an availability requirement of only 50% for the same service.

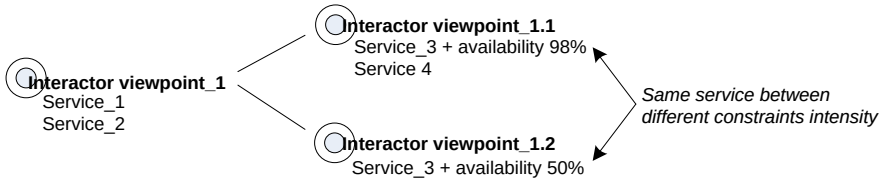


Fig. 4.5 Service and constraints variability

A viewpoint template with the following structure:

Viewpoint id	<A unique viewpoint identifier>
Type	<Viewpoint type (e.g. operator, system, component, organisation, regulatory etc.)>
Role	<Role of the viewpoint in the system>
Requirements	<Set of requirements generated by the viewpoint>

A requirement template has the following structure:

Requirement id	<Requirement identifier>
Rationale	<Justification for requirement>
Description	<Natural language definition>|<Service description>| <Other description>

Requirements can be considered at different levels of abstraction to allow for scoping and are ranked according to the benefit they offer [Kotonya04a] as follows:

- Essential (3): This means that the requirement is crucial, if they are to adequately deliver commitments made on them by operators and stakeholders.
- Important (2): This means that the requirement may prove extremely useful in assisting and delivering its commitments.
- Useful (1): This means the requirement could prove useful but it is far more likely to only be of use to a subset of operators and stakeholders.

During architecture analysis, requirement viewpoints can be used verify that proposed changes do not adversely affect critical system functionality.

- *Analysis viewpoints* allow the stakeholders involved system design and implementation to verify how well the architecture supports aspects of the system that interest them. Analysis viewpoints are associated with analysis scenarios that all the system designer to explore different architectural configurations, quality trade-offs and component solutions. We have identified three analysis viewpoints; the architect, programmer and stakeholder. The architect performs the analysis to identify critical system qualities). The result of the analysis helps the architect to propose architectural refinements that match the desired system qualities as closely as possible. Programmer is concerned with ensuring the runtime composition is structurally consistent while minimising changes that might adversely affect critical system qualities. Stakeholders are decision makers who are responsible for the project investment and domain experts who are knowledgeable in application domain.

4.1.2 Architecture Parsing

CSAFE supports two types of architecture description. The first uses UML to model the system architecture. UML is an extensible general-purpose language for modelling software systems. UML has been widely adopted by researchers and industry despite contentions over its use in modelling architecture [Medvidovic02]. UML extensions such as constraints, tagged values and stereotypes are used to extend the semantics of UML modelling elements and to define UML modelling elements with new semantics. However, many software architectures are still typically described using an architecture description language (ADL) [Medvidovic00].

Many ADLs have been developed by academic and industrial communities, including C2 [Medvidovic96], Acme [Garlan97], Darwin [Darwin95], Rapide [Luckham95], xADL [Dashofy02], and others. Each of these vary in the modelling notation used, the kinds of entities they describe, the properties they express about the entities, and how the entities may be connected (e.g. C2 is used for highly distributed software systems). However, to support independent architecture analysis that is not tied to a particular language or methodology, we developed an integrated architecture description language based on the markup language, XML [Xml10], called iXML. iXML builds on xADL and extends it to support the notion of services, non-functional requirements (e.g. constraint and its details) and inclusiveness of interface contracts (e.g. property and constraint).

4.1.2.1 Constructing Baseline System Architecture

A CSAFE baseline architecture is constructed by partitioning service descriptions and their associated constraints into abstract component (i.e. design-time components). The mapping process is aided by the CSAFE toolset. Fig. 4.6 shows the graphical process of mapping requirements to abstract components.

The process offers several advantages without compromising the architecture analysis process. These include; development traceability from requirements to deployment, process documentation, flexible implementation (i.e. abstract components can be treated as placeholders for custom development), easy mapping of abstract components to UML component notation, and a framework for change impact analysis. The process is discussed in more detail as part of the CSAFE evaluation, in Chapter 5 and Chapter 6.

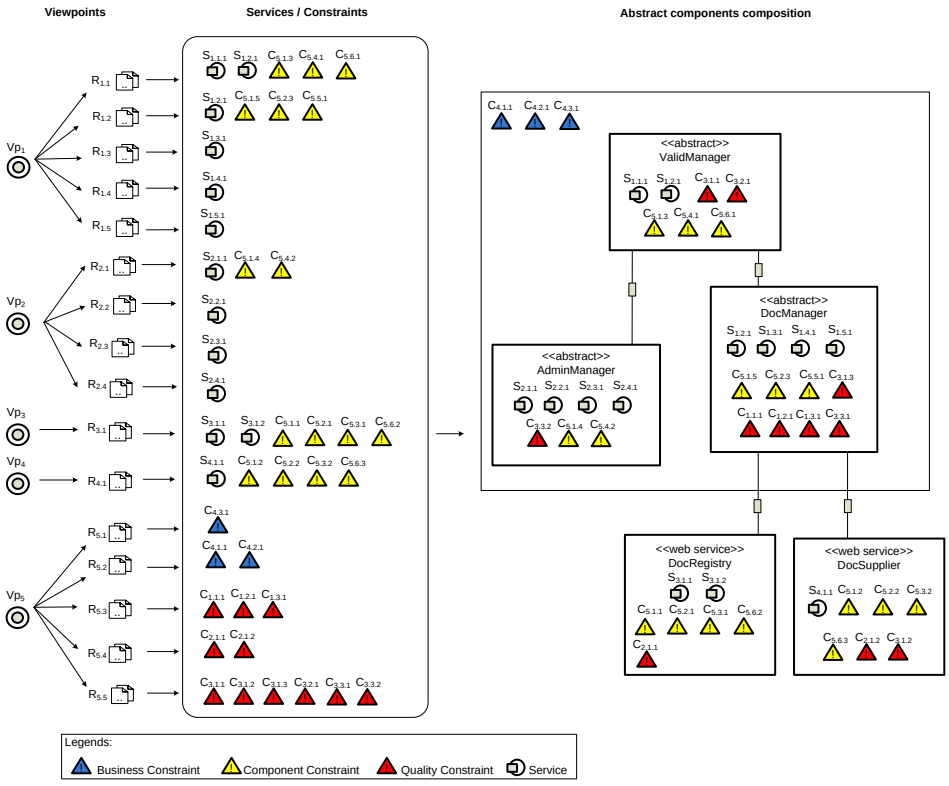


Fig. 4.6 Service partitioning

4.1.2.2 XMI/XML Parser

The XMI/XML parser supports the early stage of CSAFE analysis by parsing architecture specified in the UML notation or iXML ADL as illustrated in Fig. 4.7. Architectures specified in UML are transformed to iXML ADL, whilst architectures specified in iXML ADL are verified for correctness using the XML schema described

in Appendix A1, Table A1.1. The parser incorporates semantic safeguards to verify that components are properly connected and to the right components. The parses process outputs are stored in the analysis repository. The system architecture, architectural design templates and components specifications are all represented in the same way using a standard XML schema.

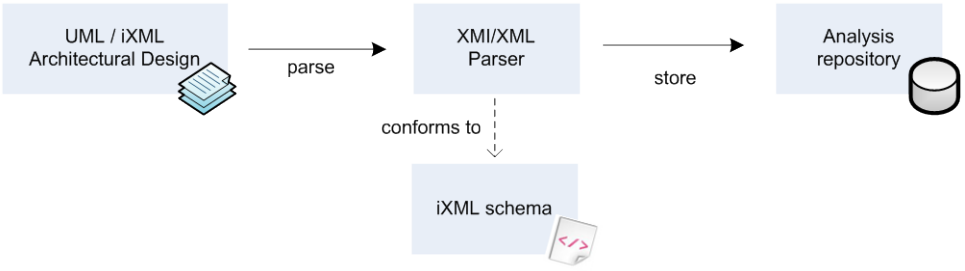


Fig. 4.7 Process parsing and storing XMI/iXML specification

The XML schemas define the structures of architecture designs, design templates and component specifications. For example, the XML schema for an architecture design specifies the elements in the design specification, nested elements, attributes of the elements, attribute values and value types. On the other hand the XML schema for a design template may also specify the elements of the design template, category, intent, context, motivation etc. The parser provides a uniform interface to the underlying XMI/XML objects. This uniform representation facilitates easy retrieval of different elements of the architecture design.

To illustrate the transformation process, consider the example of the UML architecture description of an Electronic Document Delivery and Interchange System (EDDIS) shown in Fig. 4.8. The complete system is discussed in detail as a case study in Chapter 5.

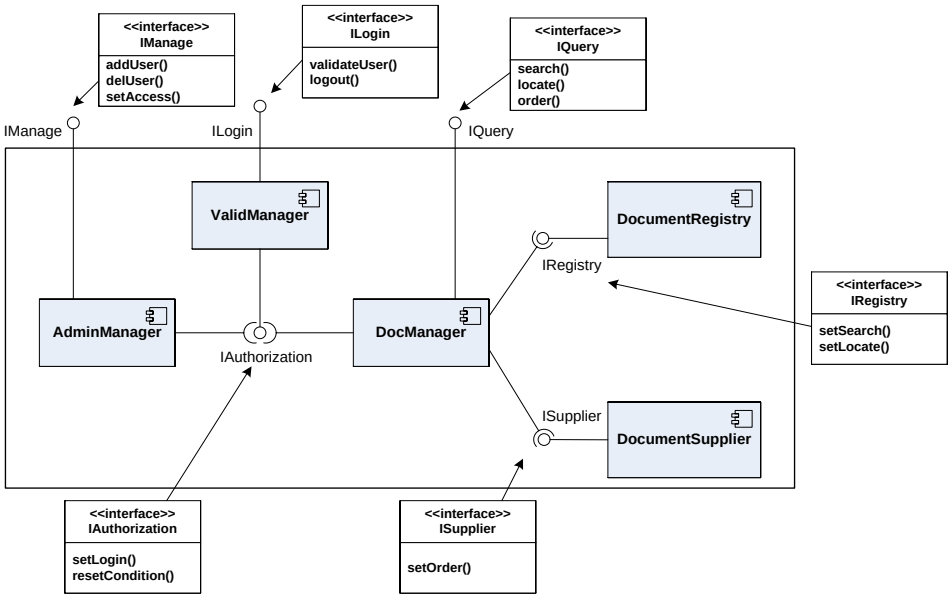


Fig. 4.8 EDDIS architectural description with interface identification

The *DocManager* component is responsible for coordinating and managing the order and delivery of electronic documents from suppliers. It has four interfaces to facilitate these services; *IAuthorisation* for accepting orders and validating recipients, *IRegistry* for finding document identifiers and their locations, *IQuery* for searching and locating documents and *ISupplier* for interacting with document suppliers. In addition, the *DocManager* component may have several properties and constraints as shown in Table 4.1. This information is part of the UML architecture description of EDDIS.

Table 4.1 *DocManager* component specification

Name	DocManager
Type:Subtype	Component
Description	Users will have access to a set of services determined by the permissions associated with their account. All users are allows for document search and locate. Only staff library can place document order. A document search will be initiated by a search criterion. The output will be a set of document identifiers. A document locate service will be initiated by a set of document identifiers and the output shall be asset of location identifiers.
Properties	- Component.Standard = null - Component.Cost = null - Component.Version = 0.2 - Component.Availability = inhouse

	- Component.Certification = No - Component(In) = 4 - Component(Out) = 2 - Component.Services = IDiscovery, IOrder - Business.Cost = Null - Business.Schedule = Null - Business.Platform = Windows XP - Reliability.Availability = Nul - Maintainability.Time = Null - Maintainability.Requirement = user - Maintainability.Technology = Null - Performance.ResponseTime_UPL = 0.5 sec. - Performance.ResponseTime_PL = 3 sec. - Performance.Throughput_UPL = 150 trans. per sec. - Performance.Throughput_PL = 75 trans. per sec.
Constraints	- Performance of response time must less than or equals to 0.75 sec. under-peak-load and less than or equals to 4 sec. peak-load. - Performance of throughput must greater or equals to 150 trans. per sec. under-peak-load and must greater or equals to 70 trans. per sec. peak-load. - Maintainability of requirement must equals to user. - Component of availability must equals to inhouse. - Business of platform must equals to Windows XP.
Interfaces	Provided -> IDiscovery, IOrder Required -> IRegistry, ISupplier, ILogin

Fig. 4.9 shows a snippet of the resulting XMI specification of the *DocManager* component with its associated constraints and textual descriptions after parsing. The XMI specification includes components, their interfaces, properties, interconnections, constraints and textual descriptions. The specification is stored in the analysis repository.

```
<Component xmi.id="Im456fe435m1254d641e78mm7be8" name="DocManager" visibility="private"
isSpecification="false" isRoot="false" isLeaf="false" isAbstract="false" isActive="false">
<ModelElement.constraint>
<Constraint xmi.idref="I3003240am1254ec16e03mm7db6"/>
 <Constraint xmi.idref="I3003240am1254ec16e03mm7daa"/>
 <Constraint xmi.idref="Im7e3cc993m12665521f35mm7b27"/>
 <Constraint xmi.idref="Im76dd02a2m12668f792b5mm7bef"/>
 <Constraint xmi.idref="Im76dd02a2m12668f792b5mm7bed"/>
</ModelElement.constraint>
<ModelElement.taggedValue>
 <TaggedValue.dataValue>Users will have access to a set of services determined by the permissions
associated with their accounts. All users are allows for document search and locate. Only library staff
can place document order. A documents locate service will be initiated by a set of document
identifiers and the output shall be a set of location identifiers. A document search will initiated by a
search criterion and a list of databases to be searched. The output will be a set of document
identifiers.
```

```
</TaggedValue.dataValue>
….
<ModelElement.taggedValue>
…..
</Component>
```

Fig. 4.9 XMI/XML specification of *DocManager*

4.1.2.3 CSAFE Architecture Description Language – iXML

The iXML ADL defines three primary architectural elements; component, interface, and connector. In addition, the iXML ADL also defines property and constraint elements that may be associated with the primary architectural elements. Fig. 4.10 shows the meta-model that provides definitions for iXML elements. The descriptions of these elements are as following:

- *Component.* Denotes an encapsulated, distributable, and executable piece of software that provides and receives services through well-defined interfaces.

  - A component has a name @ identifiers (e.g. Order)
  - A component has a stereotype (e.g. infrastructure, database, UI, web services etc.) and visibility (e.g. private, public).
  - A component may have a textual description of the component (e.g. Order component handles customer's order that include create order, search and display information).
  - A component may have one or more constraints (e.g. Order component can only be connected to EJB components).
  - A component may have one or more properties (e.g. Order component is version 0.2)
  - A component may have one or more interfaces of provided and required (e.g. Order: OrderEntry, OrderableItem and Person). It is not necessary for all of provided and required interfaces to be occupied.
  - Components that are grouped together in 'container' component form a composite component.

- *Interface*. Defines a collection of one or more operations without their implementation details. An interface can be either provided (i.e. characterizes the services that the component offers to its environment) or required (i.e. characterizes the services that the component expects from its environment).

- Interface has a name @ identifier (e.g. IOrderEntry)
- Interface has a stereotype and visibility (e.g. public, private)
- An interface may have a description (e.g. IOrderEntry is Order component's provided interface that consists of three services of CreateOrder, AddOrder and ValidateDetails).
- A provided interface may provide one or more services.
- Interface may have one or more signatures that describe operations and their attributes (e.g. IOrderEntry: AddOrder, AddOrder(item <int> , quantity <int>, total <int>)).
- An interface may have one or more constraints. A constraint can be either associated with a pre-condition or post-condition that describes restriction that must be fulfilled before and after connections to the interface.
- An interface may have one or more properties (e.g. IDoc interface is using standard Z39.50).

- *Connector*: Denotes the connection between two interfaces that defines that one interface provides the services and that the other interface requires the services.

- A connector has a name @ identifier (e.g. Order->Customer).
- A connector has a stereotype (e.g. HTTP, TCP/IP, RPC, Database Connector etc.) and role (e.g. Listener, Writer etc.)
- A connector may have a description (e.g. Order->Customer RPC feature TCP transport (RFC 793) provides a reliable and stateful connection).
- A connector may have one or more constraints (e.g. Order component communicates with Customer component must be connected via RPC).
- A connector may have one or more properties (e.g. Order->Customer is using RPC 793).
- Connector implicitly describes interconnection between two components.

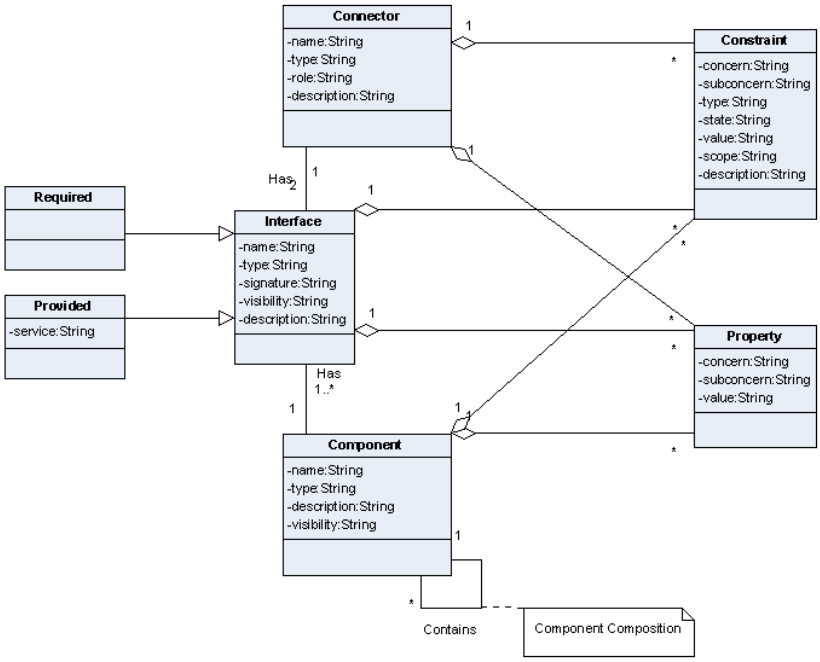


Fig. 4.10 iXML architecture meta-model

- *Constraints*. Correspond to non-functional requirements such as component cost, standard, certification and platform restrictions, or dependability requirements such as security, performance and reliability. A constraint may associate with a component, an interface, a connector or a configuration.

o A constraint has a concern (e.g. Component) and its sub-concern (e.g. Standard).
o A connector may have a description (e.g. Document delivery service shall conforms to Z39.50 document retrieval standard.).
o A constraint has a type (e.g. pre-condition, post-condition or invariant).
o A constraint has a state that indicates the state of a property or variable (i.e.. equals (EL), not equals (NE), greater than or equals (GE), greater than (GT), less than (LT) or less than or equals (LE)).
o A constraint has a value (e.g. Z39.50)
o A constraint has a scope (e.g. Identifier of service affected by the constraint)

- *Property*. Are used to extend the specification of the element by defining additional attributes that apply to architectural elements.

o A property has a concern (e.g. Performance) and its sub-concern (e.g. Response time).
o A property has a value (e.g. 4 seconds)

The iXML ADL inherits XML's schema-based extensibility mechanism allowing its rules to be extended to support specific needs. Thus, an extension may be written to modify the elements that we have described above. As indicated in section 4.1.1, the iXML ADL also supports the derivation of architectures from viewpoint requirements (i.e. services and constraints).

4.1.3 Formulating Analysis Scenarios

Analysis scenarios are formulated after architectural transformation has taken place. Analysis scenarios allow software designers and other system stakeholder to tailor the analysis to explore how specific system concerns may be addressed. Analysis scenarios provide system stakeholders with a means to augment architectural descriptions with specific quality concerns and other architectural information as part of the analysis. Designers can also formulate scenarios to explore "what if" analysis such as assessing the impact of change and competing designs. Table 4.2 shows the elements of an analysis scenario.

Table 4.2 Scenario formulation template

Aspect	Description
Concern	A desired quality attribute that acts as goal to be addressed and achieved during the process of architectural design. Concerns are associated with user requirements, component expectations and business concerns. Concerns may be categorized as follows: • Requirement (e.g. performance, security, efficiency availability, maintainability), • Component (e.g. certification, standards, resources etc.) • Business (e.g. nature of support, trust, cost)
Sub-concern	A lower level of concern that allows either qualitative or quantitative measurement to be conducted.
Refinement	Refinement expresses concern/sub-concern in more detail. For example, a broad goal such as "modifiability" or "high throughput" is not specific enough information to assess the suitability of a software architecture. A refinement is expressed as : *Concern(Sub-concern) <relational operator> <value> unit*
Conformity condition	A condition that must be satisfied in order to ensure conformity to constraint or design heuristic. Conformity conditions expressed using: • Precondition – a condition that must be true before the associated scope is executed. • Postcondition – a condition that must be true after the associated scope is executed. • Invariant – a condition that must always evaluate to be true.
Scope	Identifies services or components affected by a concern/sub-concern. Scope also serves as a traceability mechanism by providing an understanding of interrelationship between a service or a constraint, and architectural design.
Weighting	Prioritises concerns. Values assigned to quality concerns are likely to vary with application and organization. For the purpose of the evaluation described later in this thesis, I have adopted a 3-level weighting scheme that relates the value of required features to customer satisfaction and system operation. The weighting scheme of High (H), Medium (M) and Low (L) is associated with quantitative values of 3, 2 and 1: • High denotes core quality concerns. Failure to provide these features means the system will not meet customer needs. • Medium denotes features that are important to the effectiveness and efficiency of the system. Lack of inclusion of an important feature may affect customer or user satisfaction. • Low denotes features that are useful but not central to the system operation. However, lack of inclusion of a useful feature will not have significant impact on customer satisfaction.

Table 4.3 shows part of a typical scenario is formulation with concerns, sub-concerns, and their refinement, type, weighting and the concern scope.

Table 4.3 Scenario descriptions

Concern	Sub-concern	Description (Refinement)	Wt.	Scope
Component	Availability	Component(Availability) equals to web service	High	accessLocate
Component	Cost	Component(Cost) less than to 500	High	accessLocate
Component	Availability	Component(Availability) equals to web service	High	accessOrder
Component	Version	Component(Version) greater than or equals to 0.3	Low	admin_ services
Component	Certification	Component(Certification) equals to yes	High	user_ validation
Component	Version	Component(Version) equals to 4.0	Medium	user_ validation
Maintainability	Technology	Maintainability(Technology) equals to updated	Medium	user_ validation
Maintainability	Time	Maintainability(Time) less than or equals to 12 months	Medium	user_ validation
Business	Platform	Business(Platform) equals to Windows 2000/XP	High	System
Business	Schedule	Business(Schedule) equals to strict	High	System
Performance	Response Time_PL	Performance(ResponseTime_UPL) less than or equals to 0.75 seconds	High	System
Performance	Response Time_UPL	Performance(ResponseTime_PL) less than or equals to 4 seconds	High	System
Performance	Throughput_ PL	Performance(Throughput_PL) greater than or equals to 150 transaction/per second	Medium	System

4.1.4 Analysis

The analysis process is based on a flexible XML framework that allows the system designer to integrate different analysis methods and tools (see Fig. 4.14). The tools are used to check and suggest improvements to various aspects a software architecture at design-time (i.e. mapping of services to design templates) and at compose-time (i.e. mapping of abstract components to concrete components). Currently the analysis process provides support for:

- *Structure checking*: Identifies mismatches between provided and required interfaces and defects in dynamic component interaction.

- *Quality checking*. Identifies inconsistencies and mismatches between desired quality attributes (dependability, organisational, component, etc.) and the system context.
- *Conformance checking*. Verifies architectural adherence to design heuristics and styles

A typical analysis process begins with the mapping of analysis scenarios onto a repository of architecture design templates as shown in Fig. 4.11. The aim of the mapping process is to identify design templates whose contribution to specific quality attributes match the quality thresholds identified in the analysis scenarios. The quality scenarios generate query expressions that are combined with a set of rules to search the repository for design templates that match their quality thresholds. The output of the mapping process is a set of recommended design templates. The analysis process rates each recommendation on how well it contributes to the quality concerns identified in an analysis scenario.

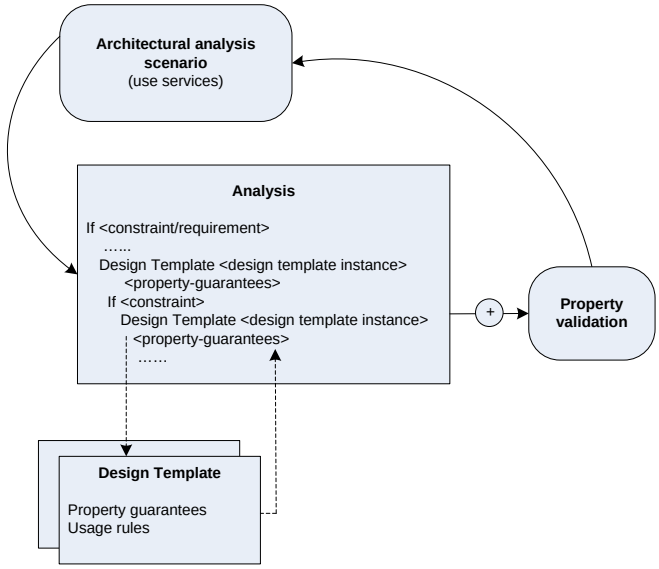


Fig. 4.11 Analysis of design mapping

Architecture design templates are uniformly specified in XML for ease of matching and to promote portability. Fig. 4.12 shows the elements of an architecture design template.

{*Category*} Type, i.e. style, design pattern, local scheme.
{*Name*} Denotes unique design template name.
{*Also-Known-As*} Other well-known names for the design template if any.
{*Related-Template*} Reference to other closely related design templates.
{*Intent*} The justification for design template
{*Context*} The situation in which the template may apply.
{*Motivation*} Describes template solution.
{Configuration} Specification of the template.
{Consequences (Contribution)} Specification of dependency and contribution that template may possess shown in scoring factor:

- High – Strongly supported,
- Medium – Moderately supported
- Low – Weakly supported.

Fig. 4.12 Architecture design template

Architecture design templates have three major benefits. First, they help in understanding and predicting the properties of design by offering a context for the creation and application of design experience. Secondly, they reduce the effort needed to understand another person's design by reducing the number of new concepts to be learned. Thirdly, they aid in creating and documenting a system design by providing rationale for component composition.

The next stage in the analysis involves modifying the system architecture to take into account the proposed recommendations. This is a two-stage process:

- First, the current system services are mapped onto the recommended architecture design templates. This activity must take into any specified constraints and design heuristics.
- Secondly, abstract components in the selected alternative architecture are mapped onto concrete components.

The process of mapping system services onto architecture design templates is tool-supported and involves selecting the relevant source component service and searching within the design template for a matching service. When a matching service is found, the destination component name appears and the service mapping is completed. The process takes into account the dependencies between different services and the

component interfaces. Fig. 4.13 shows a screenshot of the process in action where a service called *document_services* is found provided by *IRequest* interface of *DocumentRequesterB* and the service is subsequently mapped on the destination component.

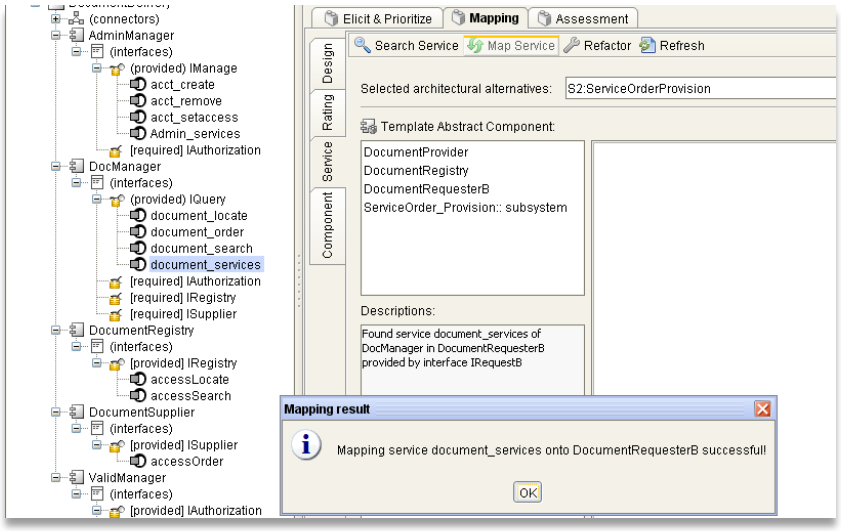


Fig. 4.13 Mapping a service onto a design template component

In cases where a matching service cannot be found, the system designer may map the service manually using a re-factoring facility provided by the tool (see Fig. 4.14). Re-factoring also allows the system designer to configure connectors, and instantiate required and provided interfaces for the destination component while ensuring that specified constraints are not violated.

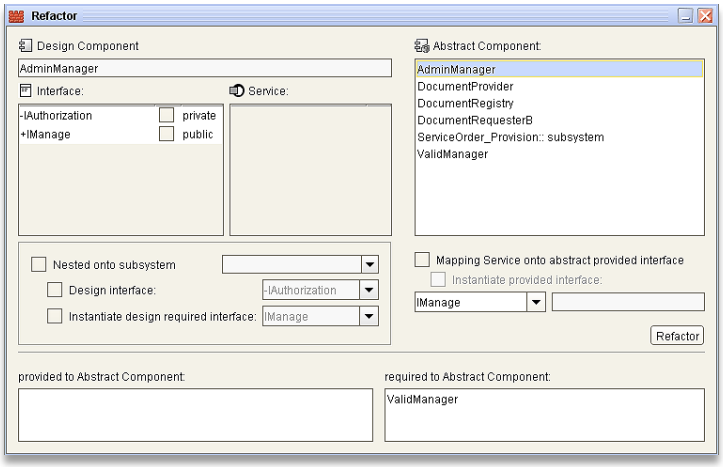


Fig. 4.14 Re-factoring facility menu

The algorithm of mapping services onto the recommended architecture design templates are as follow:

```
Algorithm DesignTemplate<design template instance><property-guarantee>
Begin
 For each service do
   search designTemplate(service)
   If service = found then
     check interface.signature
      If signature match then
        For service.constraint[ ] do
          propertyValidation(constraint)
          match[true/false]
        End For
        If match = true then
          map service → dest.component
        End If
        Else If match = false then
          flag mismatches
        End Else if
      End If
   End If
 End For
End
```

The second stage of the process involves mapping abstract components to concrete components. The analysis tool aids the process by indicating how well the mapping fits, exposing mismatches and providing suggestions for further component selection. Fig. 4.15 shows how the tool supports the process of mapping of an abstract component, *AdminManager*, to three concrete components (i.e. *AdminManager_1, AdminManager_2 and AdminManager_3*). Concrete component fitness is indicated in percentage terms next to the concrete components. The mismatches associated with each concrete component are also indicated.

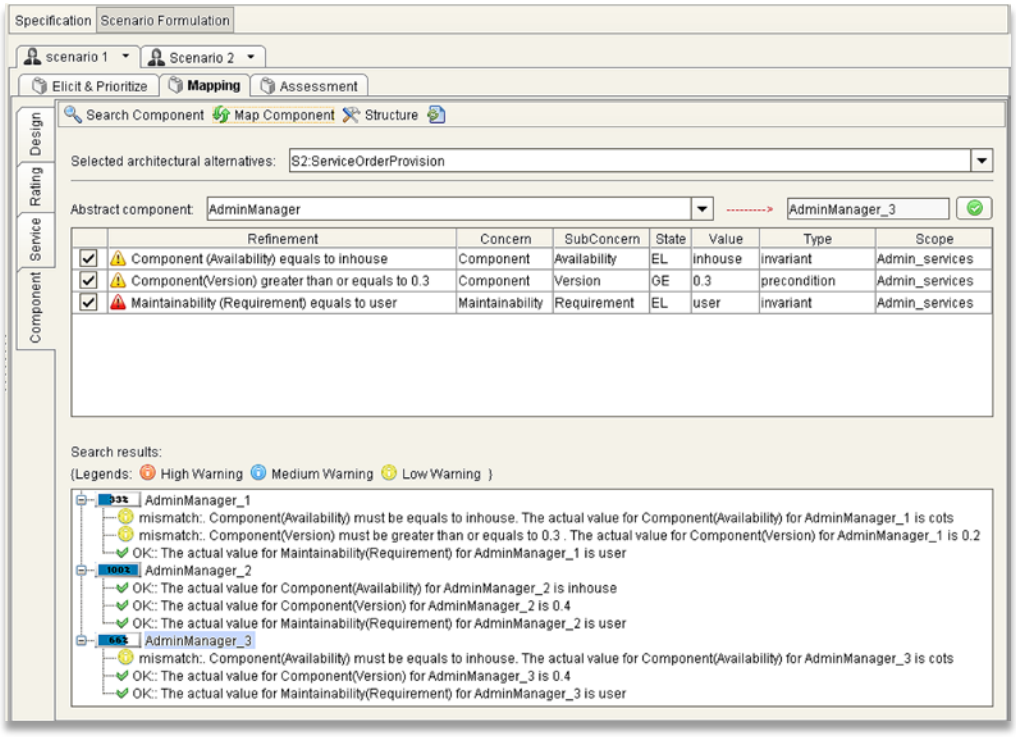


Fig 4.15 Mapping onto concrete component

Like architecture design templates, concrete component are also uniformly specified in XML as shown in Table 4.4. Detailed specifications for the concrete components used in this research are provided in Appendix A3.

Table 4.4 Component template

{Name} denotes a unique name for the component
{Type:Subtype} denotes the component with a particular behaviour and services its deliver
{Description} denotes a details explanation of the component
{Properties} denotes component's concern and sub-concern and its values
{Constraints} a predicate imposed on one or more component properties
{Interfaces} denotes the interfaces specified on the component

4.1.5 Trade-off Analysis and Rating - Negotiation

Trade-off analysis is intended to support the process of balancing the architectural considerations and stakeholder concerns with the available component functionality. CSAFE supports trade-off analysis through the implementation of the *Simple Multi-*

Attribute Rating Technique (SMART) [Shepetukha01]. SMART is a form of the multi-attribute utility theory methods.

Although SMART has some similarities with other multi-attribute analysis approaches such as the *Analytic Hierarchical Process (AHP)* [Saaty90], it does have its own peculiarities. As with AHP, SMART contextualises the decision making process to a decision maker and a set of previously identified options to be considered. Rather than relying on pair-wise comparisons, an assumption of the SMART approach is that performance against attributes can ultimately be measured, and a value assigned. Although various mechanisms can be used to measure performance and assign values, where appropriate, value functions can be used. This ability to capture a subtle, perhaps subjective and possibly complex relationship between an option's performance according to a particular attribute and the value assigned to that performance is a potential strength of SMART. Another is the weighting of the attributes in order to recognise their relative priority. Together, these aspects allow SMART to balance different strengths and weaknesses across options, and allow for a degree of weighted trade-off.

SMART provides a means for assessing each of the quality concerns to reflect its relative importance to the design decision**.** By refining the scores with the relative weights of all quality concerns, the utility value or contribution for each alternative solution can be computed. The utility function used in SMART [Shepetukha01] is shown below; where w_i is the scaling value (weight) assigned to the ith of m quality concerns, s_{ij}^* is the utility for alternative j on criterion i, and n is number of alternative solutions.

$$\mu_j \quad = \quad \sum_{i=1}^{m} w_i \, s_{ij}^* \Big/ \sum_{i=1}^{m} w_i \ , \qquad j = 1 \ldots n,$$

A maximum score of 1 for the utility value indicates the highest probability of the quality concerns being achieved. Whereas, the minimum of zero indicates the least

acceptable trade-off. While high rating scores increase the likelihood of an architecture design template being selected, specific analysis of its contribution to individual quality concerns may be needed to provide better understanding of the results at every level. Fig. 4.16 shows the example of three alternative architecture contributions to different quality sub-concerns, generated by the CSAFE trade-off analysis.

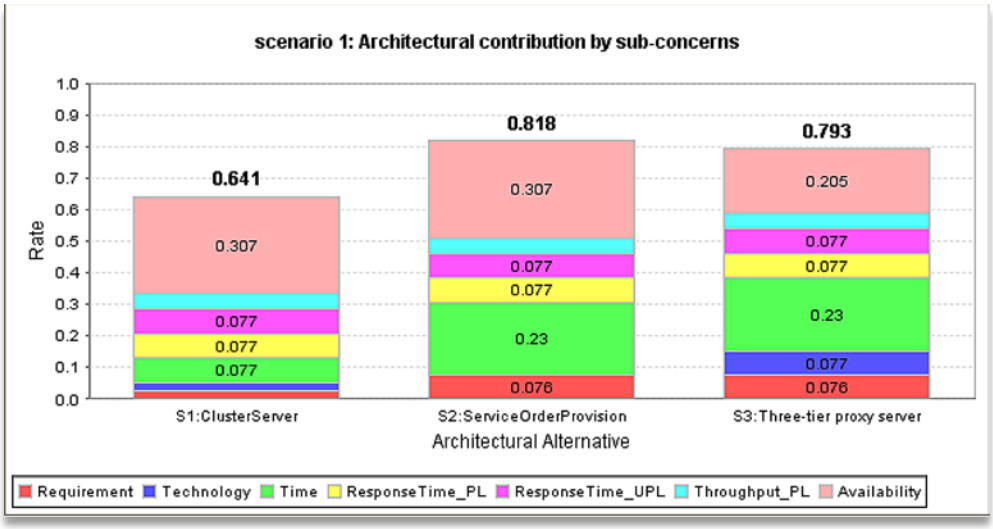


Fig. 4.16 Contribution of suggested alternatives according to sub-concerns

In addition to trade-off analysis, CSAFE provides support for sensitivity at quality concern and sub-concern levels. Sensitivity analysis may be needed to establish how robust the choice of an architecture is to changes in the weights for quality concerns identified in the analysis scenario. Conducting sensitivity analysis can help the system designer understand how variations in the relative weights of critical quality concerns might affect the suggested solutions, and may lead the designer to reconsider some of the weights associated with the quality concerns. Sensitivity analysis shows that, in many cases, large variations in the weights are often required before one option becomes more attractive than another. It is therefore possible, in certain cases, to trade-off quality concern weights without adversely affecting the system quality.

The system designer performs sensitivity analysis by making systematic changes to the relative weights of the quality concerns and observes how the variations affect the contributions of the recommended solutions. Changes may involve:

- Varying concern (q) weights to minimum one at a time:

$$\mu_j = \sum_{i=1}^{m!=q} w_i s_{ij}^* \Big/ \sum_{i=1}^{m} w_i , \quad j = 1 \ldots n, \ where\ w_q = 0$$

- Varying concern (q) weights to maximum one at time:

$$\mu_j = \sum_{i=1}^{m=q} w_i s_{ij}^* \Big/ \sum_{i=1}^{m} w_i , \quad j = 1 \ldots n, \ where\ w_{!q} = 0$$

The results of the sensitivity analysis are a set of recommendations that comprise change impact graphs and recommendations that guide the system designer to improve the architecture design. Fig. 4.17 shows an example of a CSAFE sensitivity analysis for the maintainability quality concern. The graph shows how the benefits from three architectural alternatives vary with changes in the relative weighting of the quality concern. At weighting value of 0.38, the architectural alternative, S2, provides the best benefit and S1 the worst. At a weighting of 0.45, S3 provides the best benefit. However, S2 remains generally unaffected by the changes.

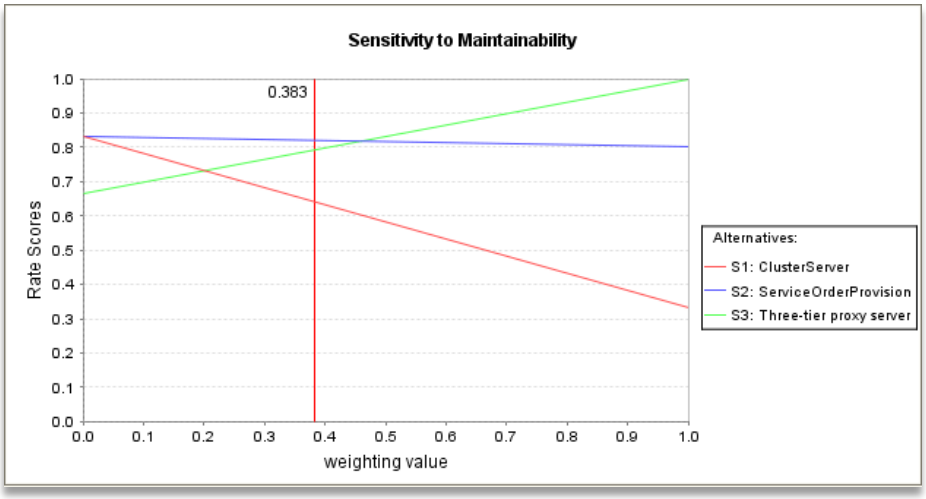


Fig. 4.17 Sensitivity analysis of Maintainability

4.2 The Toolset

CSAFE is supported by an integral toolset. The toolset was specified and designed using the UML notation, and implemented in the Java programming language. An overview of the toolset use cases is shown in Fig. 4.18. The complete use case specification and object model for the toolset is provided in Appendix B.

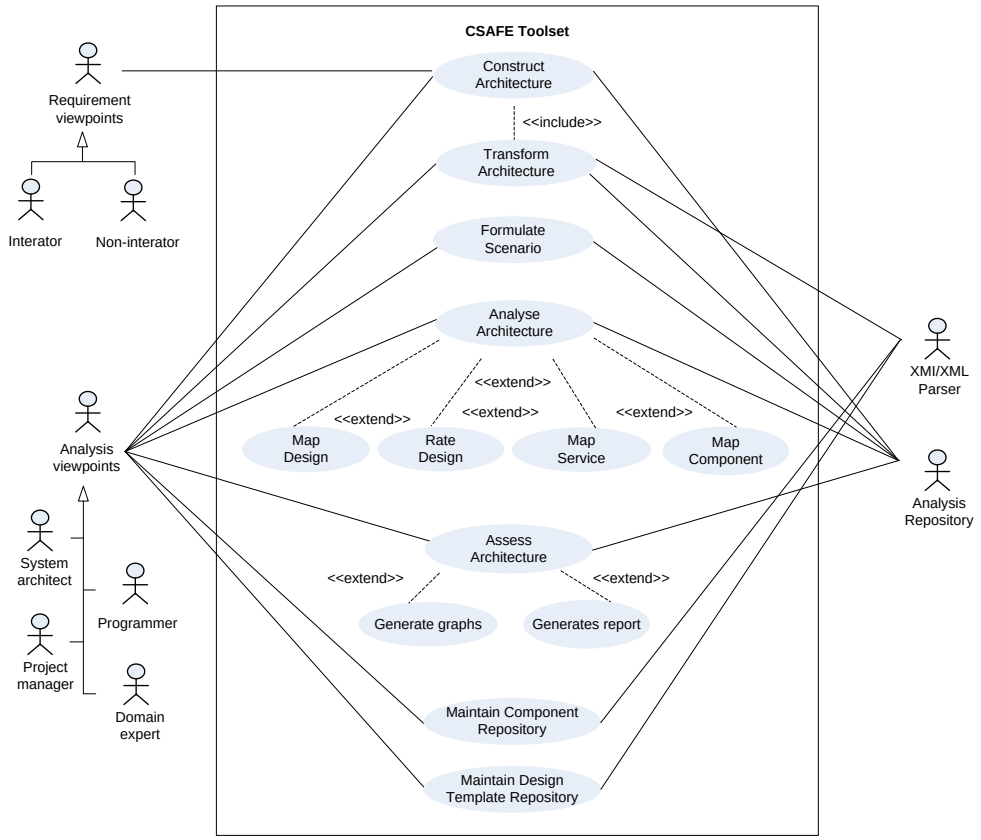


Fig. 4.18 CSAFE toolset use-case diagram

4.2.1 CSAFE Toolset Architecture

The CSAFE toolset has six main components: The XMI/XML parser, scenario formulator, analyser, iXML ADL, trade-off analyser and rater, and report generator. These components are supported by an analysis repository containing the design

template library, component library and architecture database. Fig. 4.19 shows architecture of the CSAFE toolset.

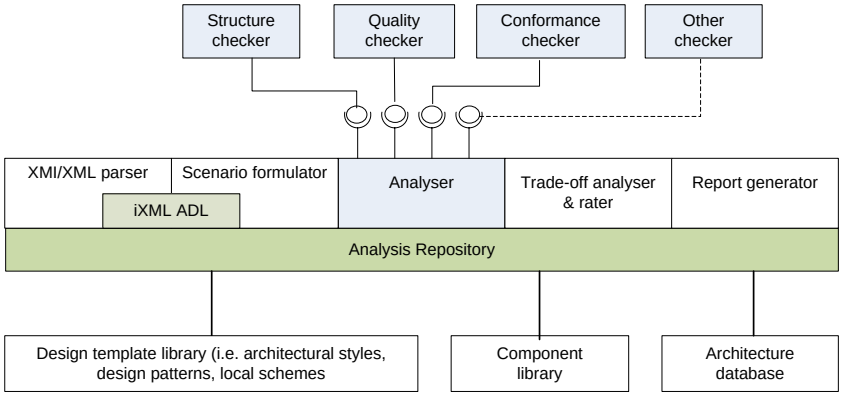


Fig. 4.19 Architecture of CSAFE toolset

XMI/XML Parser

This supports the early stage of the CSAFE process by transforming architectures expressed in UML to iXML ADL format, and by verifying architectures expressed in iXML ADL. Table 4.5 and Fig. 4.20 show the sequence of the transformation and the actors involved.

Table 4.5 Transform architecture use-case description

CSAFE: Transform Architecture	
Actors	System Designer, XMI/XML Parser, Analysis Repository
Description	1. System designer selects the XMI/XML architectural specification from the analysis repository. 2. System designer enters project name and clicks OK. 3. The XMI/XML parser parses the architectural specification and checks it against XML schema/DTD. 4. The XMI/XML parser creates a design schema for the architecture. 5. The XMI/XML parser stores the architectural vectors in analysis repository. 6. The tool organizes the architectural elements into a tree hierarchy.
Data	XMI/XML architectural specification
Stimulus	System designer selects 'New Project' from CSAFE File menu
Response	CSAFE parses and stores the architecture design in the analysis repository.
Alternative flow of events	3.a. Invalid XMI/XML description. Indicate error message.

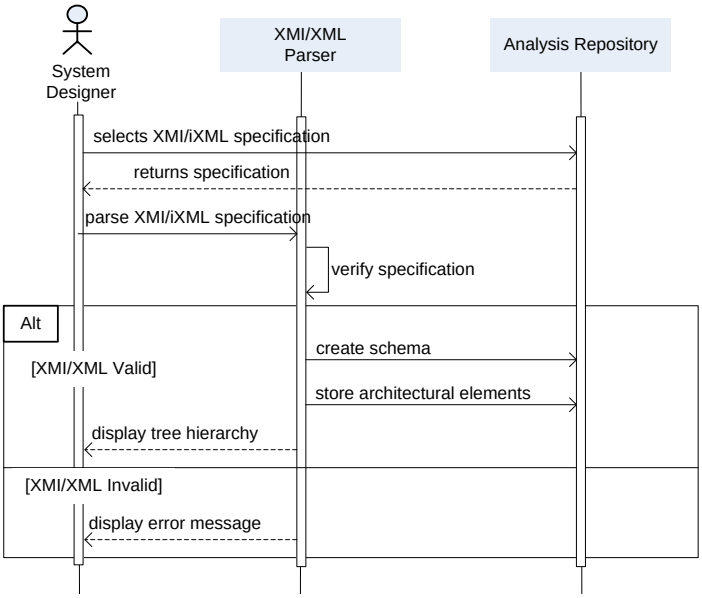


Fig. 4.20 Transform architecture sequence diagram

Scenario Formulator

The scenario formulator allows the system designer and other stakeholders to identify and explicitly represent system quality concerns as goals to be addressed and achieved during the process of architectural design. Product quality concerns can be associated with any element of the system design. To facilitate scenario formulation, the tool incorporates a process for weighting and ranking quality concerns based on the scheme described in section 4.1.3. Data from analysis scenarios provide input to the analysis and trade-off processes. A use case description of the scenario formulation process is provided in Appendix B1.2. Analysis scenarios are stored in the analysis repository.

Analyser

The analyser is responsible for mapping analysis scenarios onto architecture design templates, and for transforming abstract system designs to concrete compositions. The analyser incorporates a set of rules that relate quality concerns in analysis scenarios to design templates to identify architectural solutions that best address the quality

concerns. However, no single architectural solution can adequately address all the quality concerns raised by stakeholders; every architectural solution is a trade-off of competing quality concerns. The analyser rates each architectural solution for its contribution to critical quality concerns.

Selected design templates are instantiated to facilitate service and component mapping as discussed in Section 4.1.4. The process of mapping services to instantiated design templates takes into account any specified constraints and design heuristics. The analyser has a set of pre-defined rules to ensure the service mapping proceeds correctly. Some of the rules are shown in Table 4.6. Lastly, the abstract components in a selected alternative architecture are mapped onto concrete components. The analyser flags warning messages for structural (including configuration) and property mismatches found between the components. Analysis scenarios are stored in the analysis repository. Use case descriptions of the mapping and rating processes is provided in Appendix B1.4 – B1.7.

Table 4.6 Service mapping rules

No.	Rule Description
1.	**If** (service not found and interface's design component is provided and constraints not violated) **Then** (configured connectors between design component and abstract component)
2.	**If** (service not found and design interface is 'Required') **Then** (violation: service not provided by abstract's component)
3.	**If** (service found and interface type match) **Then** (violation: attempting to connect component's interfaces of 'Provided' -> 'Provided' or 'Required' -> 'Required')
4.	**If** (service found and interface type not matching and abstract component interface is 'Provided' and constraints not violated) **Then** (configured connectors between abstract component and design component)
5.	**If** (service found and interface type not matching and abstract component interface is 'Required' and constraints not violated) **Then** (configured connectors between design component and abstract component)

Trade-off Analyser - Negotiator

The trade-off analyser is responsible to assessing and rating competing architectural solutions for their contributions to different quality concerns and different concrete components configurations. The trade-off analyser generates results in tabular and graph format for qualitative and quantitative analysis. Use case descriptions of the assessment process are provided in Appendix B1.8 – B1.9.

Design Template Repository

The design template repository stores architectural design templates and the result of analysis and composition. Fig. 4.21 shows the design template metamodel. The repository contains facts about the design templates and rules that govern their correct use.

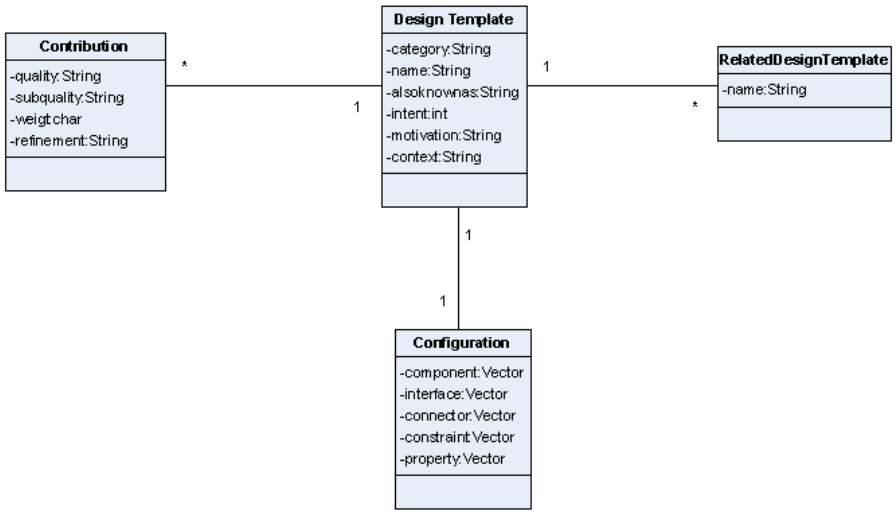


Fig. 4.21 Design template metamodel

A snippet of the design template XML Data Type Description (DTD) is shown in Table 4.7. A complete description is provided in Appendix A2, Table A2.1.

Table 4.7 Design template XML DTD description

```
<!ELEMENT NXML (CATEGORY, RNAME, ALSOKNOWNAS, RELATEDRULES, INTENT, CONTEXT,
MOTIVATION, CONTRIBUTIONS, CONFIGURATION)>

<!sELEMENT CATEGORY (#PCDATA)>
<!ELEMENT RNAME (#PCDATA)>
<!ELEMENT ALSOKNOWNAS (#PCDATA)>
<!ELEMENT RELATEDRULES (RELATEDRULE.DESCRIPTION*)>
<!ELEMENT RELATEDRULE.DESCRIPTION EMPTY>
<!ATTLIST RELATEDRULE.DESCRIPTION RNAME CDATA #REQUIRED >
<!ELEMENT INTENT (#PCDATA)>
<!ELEMENT CONTEXT (#PCDATA)>
<!ELEMENT MOTIVATION (#PCDATA)>
<!ELEMENT CONTRIBUTIONS (CONTRIBUTION.DESCRIPTION*)>
<!ELEMENT CONTRIBUTION.DESCRIPTION (#PCDATA)>
<!ATTLIST CONTRIBUTION.DESCRIPTION QUALITY CDATA #REQUIRED
                                   SUBQUALITY CDATA #REQUIRED
                                   WEIGHT CDATA #REQUIRED>
<!ELEMENT CONFIGURATION (COMPONENT*, INTERFACE*, CONNECTOR*)>
```

Component Repository

The component repository is a machine searchable library of black-box components. Most component repositories specify components using interface-description-languages (IDLs), which are restricted to describing only structural properties. Our approach uses an extensible constraint notation to express semantic properties of a component, in addition to structural properties. Constraints are expressed using concerns, sub-concerns, relational operators, conformity conditions (i.e. precondition, post-condition, or invariant), values and services. The component metamodel is shown in Fig. 4.22. A complete description is provided in Appendix A3, Table A3.1.

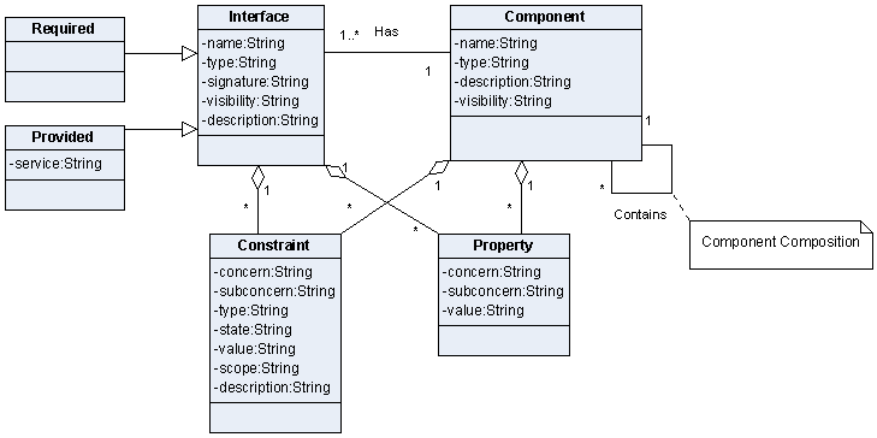


Fig. 4.22 Component metamodel

Report Generator

Report generator is used to construct reports of the architecture analysis. The report generator can be configured to generate tailored reports of the analysis to suit different stakeholder interests.

4.3 Summary

This chapter has described CSAFE, the proposed architecture analysis approach for supporting component-based black-box system development. The chapter has explained how CSAFE fits into a general design process, outlined the steps in the process and discussed each stage in the process. The chapter has explained the link between CSAFE and requirements analysis, and shown how this can be used to support initial architectural design. I have shown in this chapter how CSAFE provides supports for diverse stakeholder involvement. I have also shown how CSAFE provides explicit support for negotiation (i.e. trade-off analysis), support for standard modelling notations such as UML and diversity in analysis. The chapter has also discussed the CSAFE toolset architecture and its various components. Detailed use cases describing the functionality of the toolset are provided in Appendix B. A detailed user guide for CSAFE toolset is provided in Appendix C.

Chapter 5

Evaluation 1: Electronic Document Delivery Information System

This chapter presents the first of two case studies used to evaluate the architectural analysis framework (CSAFE) described in Chapter 4. The case study used in the evaluation is derived from the specification of an actual Electronic Document Delivery and Management System (EDDIS) [Kotonya07]. A summarised version of the evaluation has been published in [Admodisastro11]. The objective of the first evaluation is to demonstrate the key features of CSAFE and the practicability of the framework. The evaluation demonstrates how CSAFE can be used to construct, analyse and refine a software system architecture from requirements to system composition. The evaluation is conducted using two different stakeholder scenarios to demonstrate CSAFE's support for broad stakeholder involvement in architectural design and analysis.

5.1 The Case Study

The Electronic Document Delivery and Interchange Systems (EDDIS) is a web-based library system for the UK Higher Education sector to help users obtain documents, other library items not available at their local library. The main function of EDDIS is to manage the process of identifying, locating, ordering and supplying electronic documents. Users access to the system via web-based interface using valid usernames and passwords. EDDIS users have access to a range of services determined by the permissions associated with the accounts they hold. Each EDDIS node has an administrator whose task is to set up and manage user accounts.

To obtain a document, an EDDIS user must place an order with the document supplier. However, before a document order can be placed, the user must first obtain the document identifiers and its location identifiers from a centralised document registry. All document interchange between an EDDIS node and the document supplier use the Z39.50 document retrieval protocol. When the ordered document arrives on the EDDIS server it is automatically emailed to the requester as a PDF document. EDDIS users can also order non-digital items. In this case, the physical item is supplied to the library administrator who notifies the requester via email. The next section describes a subset of the EDDIS requirements and shows how the viewpoints approach described in Section 4 was used to elicit and partition them.

5.2 EDDIS Viewpoints and Requirements

The viewpoint approach described in Section 4 is used to elicit EDDIS requirements. Five viewpoints are identified for the EDDIS user (Vp_1), administrator (Vp_2), document_registry (Vp_3), document_supplier (Vp_4) and consortium (Vp_5). Table 5.1 shows the EDDIS requirements associated with each viewpoint instance. These requirements are associated with a number of services and constraints. The detailed descriptions of services and constraints are provided in Appendix D2 and Appendix

D3 respectively. Services represent expressions of required functionality expressed in way that shows the dependencies between the services.

Constraints represent stakeholder concerns such as component cost, component certification, component memory and platform restrictions, or dependability requirements such as security, performance and availability. They may also represent elements of interdependence that are introduced to allow services to meet certain architectural considerations. Finally, constraints may capture dependencies that are introduced to make certain component choices acceptable in the current context, particularly with regard to the outcome of negotiation and thus may hold important design rationale information. Each requirement is ranked as described in Section 4.1.1 (i.e. as essential, important or useful) to determine its priority level.

Table 5.1 EDDIS viewpoints and requirements

Viewpoint		Requirement			
ID	**Role/Type**	**ID**	**Description**	**Service**	**Ranking**
Vp_1	EDDIS_User (Operator)	R1.1	EDDIS users shall be able to login on to the system via a Web-based interface using valid usernames and passwords.	S1.1.1 S1.2.1	Essential
		R1.2	Once logged in, EDDIS users will have access to a set of services determined by the permissions associated with their accounts.	S1.2.1	Important
		R1.3	EDDIS shall allow users to search and identify documents, which interest them. A document search will be initiated by a search criterion and a list of databases to be searched. The output will be a set of document identifiers.	S1.3.1	Essential
		R1.4	EDDIS shall allow users to determine the location of documents. A documents locate service will be initiated by a set of document identifiers and the output shall be a set of location identifiers.	S1.4.1	Essential
		R1.5	EDDIS user shall allow users to order documents. A document order will be initiated by a set of document and location identifiers. The output will be a set of order identifiers and electronic/hardcopy documents.	S.1.5.1	Important
Vp_2	EDDIS_ Administrator (Operator)	R2.1	EDDIS shall provide facilities for setting up and managing user accounts.	S2.1.1	Important

Viewpoint		Requirement			
ID	Role/Type	ID	Description	Service	Ranking
		R2.2	EDDIS shall allow admin to create account for EDDIS user. Creating a new account require user name, matrix/staff no. and user level e.g. Undergraduate, Postgraduate and Staff.	S2.2.1	Essential
		R2.3	EDDIS shall allow admin to delete EDDIS user account. An account delete require matrix or staff no.	S2.3.1	Important
		R2.4	EDDIS shall allow admin to assign access level for EDDIS user.	S2.4.1	Essential
Vp_3	Document_ Registry (Component)	R3.1	EDDIS shall be able to access a centralized document registry to obtain document and location identifiers using the Z39.50 document retrieval standard.	S3.1.1 S3.1.2	Important
Vp_4	Document_ Supplier (Component)	R4.1	The document order client will be use the Z39.50 document retrieval standard.	S4.1.1	Important
Vp_5	EDDIS_ Consortium (Organisation)	R5.1	The system shall run on Microsoft Windows 2000 and Windows XP.		Essential
		R5.2	The system shall be develop according to schedule and cost estimated.		Important
		R5.3	The system shall ensure that a reasonable level of performance is maintained across the services at all times.		Important
		R5.4	The system shall ensure that availability of service is given to EDDIS users accordingly.		Essential
		R5.5	The system shall ensure that it is easy to maintain that allow for graceful replacements or extensions of components.		Useful

Fig. 5.1 shows the use-cases associated with high-level service descriptions that represent the underlying EDDIS functionality. These can be combined with other forms of modelling such as interaction diagrams (see Fig. 5.2) and statecharts to provide a more detailed description of the system behaviour. However, for component-based systems, detailed requirements specifications are often counter-productive as they tend preclude possible component solutions [Admodisastro06].

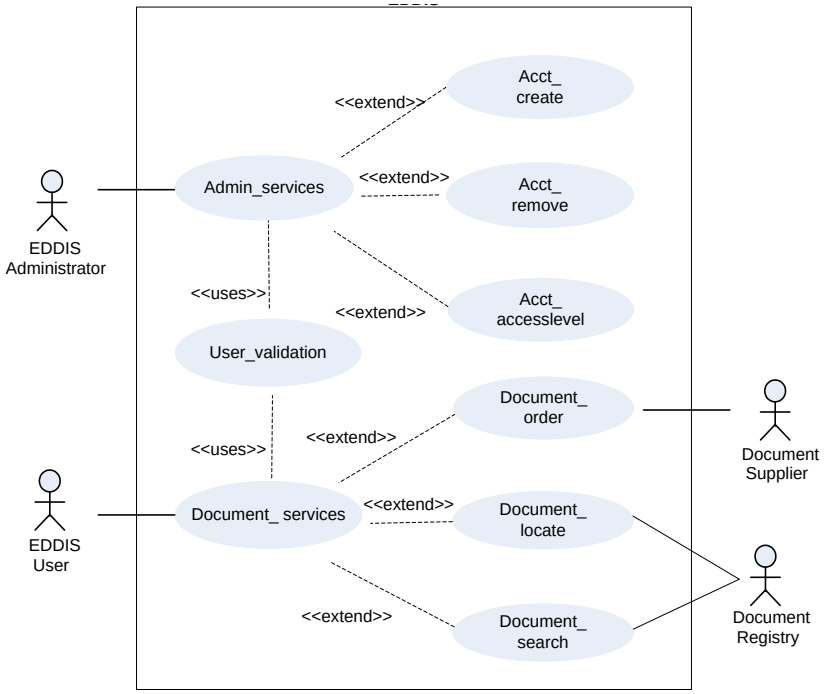


Fig. 5.1 EDDIS use-case diagram

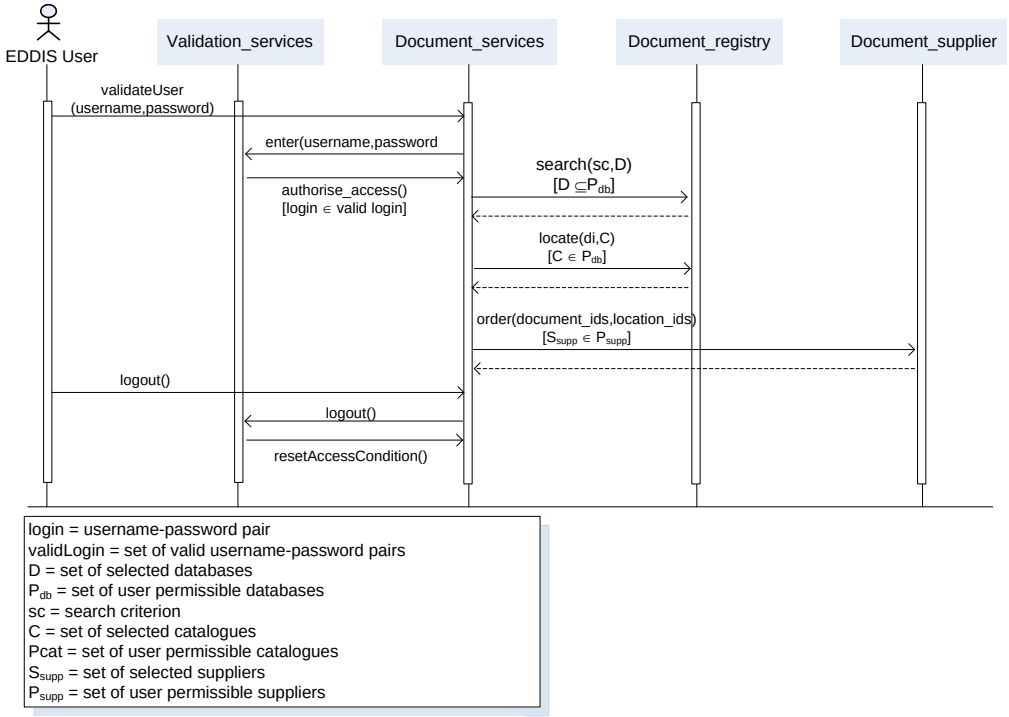


Fig. 5.2 Sequence diagram for EDDIS services

5.2.1 Constructing the baseline EDDIS Architecture

The input to the architecture analysis process is the software system architecture. A baseline architecture is constructed for the EDDIS by partitioning its service descriptions and their associated constraints into abstract components. Fig. 5.3 shows the result of partitioning of the EDDIS requirements onto five design-time components using the approach described in Section 4.1.2. The process takes into account the system and service constraints, and dependencies between the services.

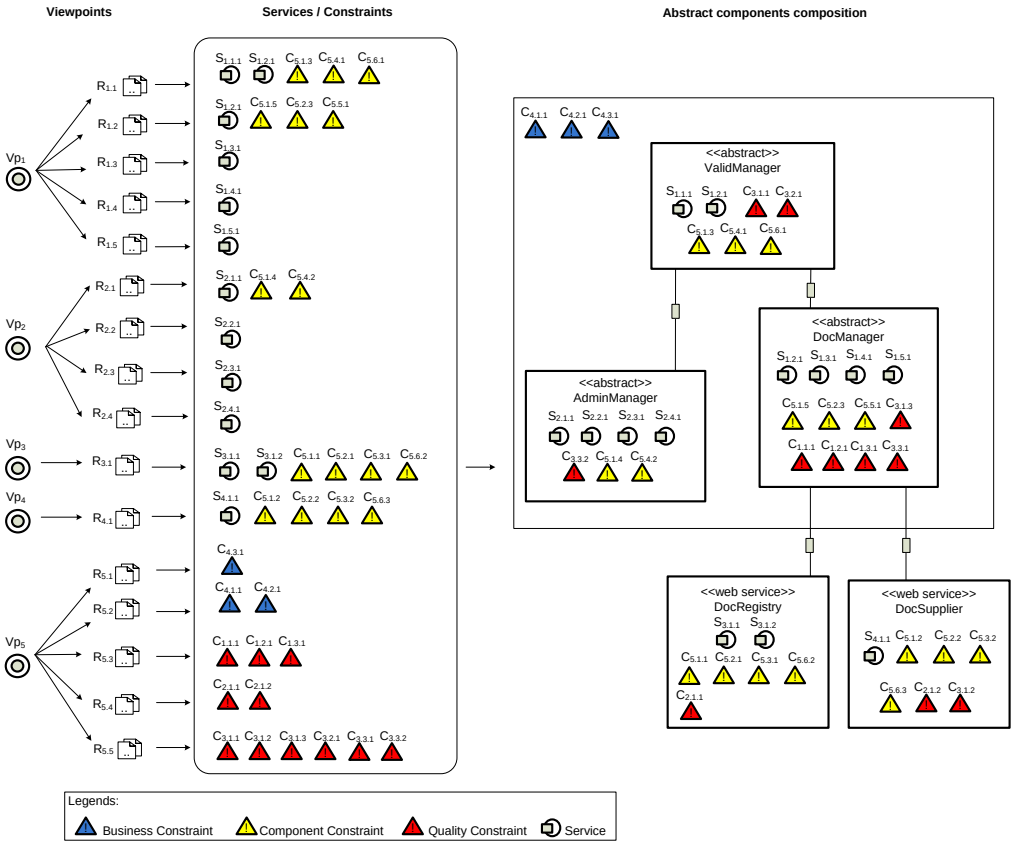


Fig. 5.3 EDDIS service partitioning

The partitioned services are then mapped onto a UML component model as shown in Fig. 5.4. In addition to enhancing the system documentation, the partitioning and mapping process provide traceability back to requirements formulation. It was decided that functionality for the *AdminManager*, *ValidManager* and *DocManager*

would be provided using off-the-shelf components while *DocumentRegistry* and *DocumentSupplier* would be provided by web services.

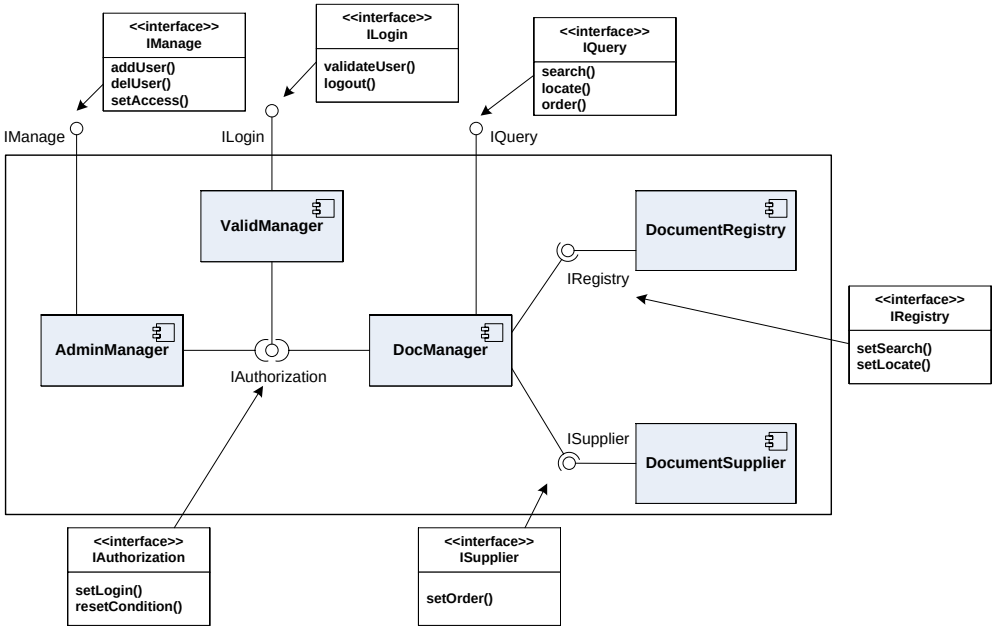


Fig. 5.4 EDDIS architectural description with interface identification

5.3 The Analysis

The architecture analysis process begins with the transformation of the UML description of EDDIS into a machine proccessable iXML specification. The CSAFE parser supports the transformation process by parsing and storing EDDIS architectural elements in an analysis repository, which is accessible by other CSAFE tools. It provides a uniform interface to the underlying XML object model that represents elements of the architecture (i.e. architectural structure with its descriptions, services, interfaces, constraints and properties). Table 5.2 shows the original *DocManager* component specification, and Fig. 5.5 shows part of the XMI/XML transformation of the *DocManager*[1] component.

[1] The description has been simplified but does not affect connotation of the content.

Table 5.2 *DocManager* component specification

Name	DocManager
Type:Subtype	Component
Description	Users will have access to a set of services determined by the permissions associated with their account. All users are allows for document search and locate. Only staff library can place document order. A document search will be initiated by a search criterion. The output will be a set of document identifiers. A document locate service will be initiated by a set of document identifiers and the output shall be asset of location identifiers.
Properties	- Component.Standard = null - Component.Cost = null - Component.Version = 0.2 - Component.Availability = inhouse - Component.Certification = No - Component(In) = 4 - Component(Out) = 2 - Component.Services = IDiscovery, IOrder - Business.Cost = Null - Business.Schedule = Null - Business.Platform = Windows XP - Reliability.Availability = Nul - Maintainability.Time = Null - Maintainability.Requirement = user - Maintainability.Technology = Null - Performance.ResponseTime_UPL = 0.5 sec. - Performance.ResponseTime_PL = 3 sec. - Performance.Throughput_UPL = 150 trans. per sec. - Performance.Throughput_PL = 75 trans. per sec.
Constraints	- Performance of response time must less than or equals to 0.75 sec. under-peak-load and less than or equals to 4 sec. peak-load. - Performance of throughput must greater or equals to 150 trans. per sec. under-peak-load and must greater or equals to 70 trans. per sec. peak-load. - Maintainability of requirement must equals to user. - Component of availability must equals to inhouse. - Business of platform must equals to Windows XP.
Interfaces	Provided -> IDiscovery, IOrder Required -> IRegistry, ISupplier, ILogin

```
<Component xmi.id="Im456fe435m1254d641e78mm7be8" name="DocManager" visibility="private" isSpecification="false" isRoot="false" isLeaf="false" isAbstract="false" isActive="false">
<ModelElement.constraint>
<Constraint xmi.idref="I3003240am1254ec16e03mm7db6"/>
 <Constraint xmi.idref="I3003240am1254ec16e03mm7daa"/>
 <Constraint xmi.idref="Im7e3cc993m12665521f35mm7b27"/>
 <Constraint xmi.idref="Im76dd02a2m12668f792b5mm7bef"/>
 <Constraint xmi.idref="Im76dd02a2m12668f792b5mm7bed"/>
 <Constraint xmi.idref="Im76dd02a2m12668f792b5mm7beb"/>
```

```
 <Constraint xmi.idref="Im76dd02a2m12668f792b5mm7bdf"/>
 <Constraint xmi.idref="Im76dd02a2m12668f792b5mm7bdd"/>
 </ModelElement.constraint>
<ModelElement.taggedValue>
 <TaggedValue.dataValue>Users will have access to a set of services determined by the
permissions associated with their accounts. All users are allows for document search and locate.
Only library staff can place document order. A documents locate service will be initiated by a set
of document identifiers and the output shall be a set of location identifiers. A document search
will initiated by a search criterion and a list of databases to be searched. The output will be a set
of document identifiers.
</TaggedValue.dataValue>
....
<ModelElement.taggedValue>
.....
</Component>
```

Fig. 5.5 XMI/XML specification of *DocManager*

Fig. 5.6 shows how the process is supported in the toolset and how the final result is organised. The root of the tree represents the overall system architecture. The nodes of the tree represent the system components. Each component has a set of interfaces and connectors. Each architectural element is also associated with an optional set of properties and constraints. Part of the XMI/XML specification of the system is shown in the right pane.

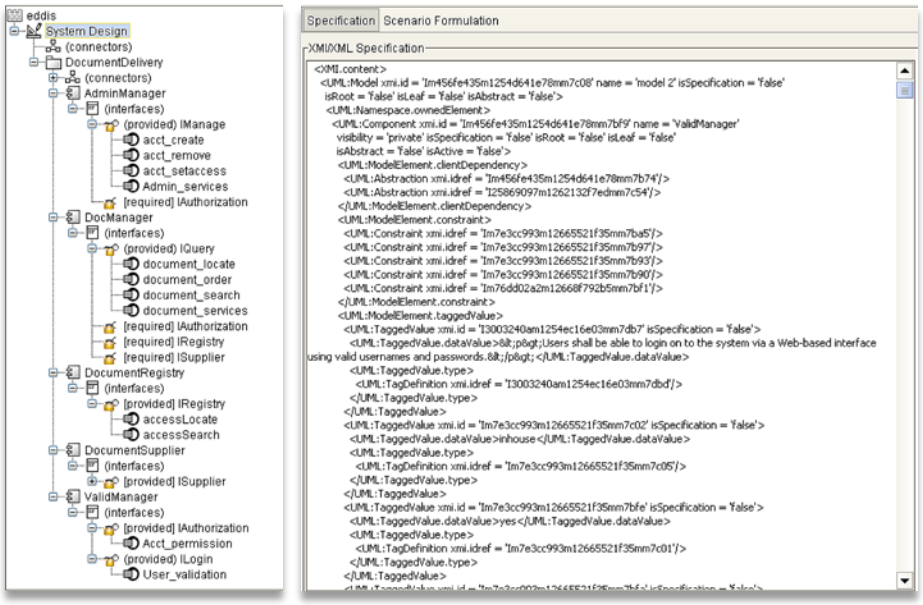


Fig. 5.6 Parsed EDDIS architecture (left pane) and EDDIS XMI/XML source file (right pane)

The EDDIS architecture tree comprises five components; *AdminManager*, *DocManager*, *DocumentRegistry*, *DocumentSupplier* and *ValidManager* (see Fig. 5.6, left pane). The architecture tree shows the component nodes expanded to reveal provided and required interfaces. A provided interface is associated with service(s), for example, *IManage* provides services; *acct_create, acct_remove, acct_setAccess,* and *admin_services*. Lastly, the parser also captures other architectural element information such as type, description, signatures, role, properties and constraints as shown in Fig. 5.7.

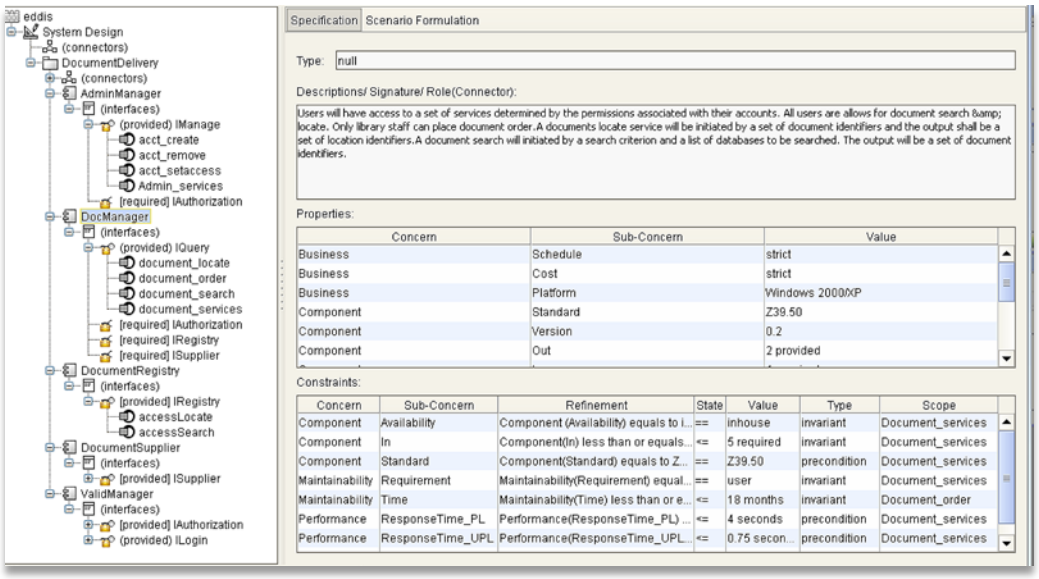


Fig. 5.7 *DocManager* component specification (right pane)

5.3.1 Formulating EDDIS Analysis Scenarios

After architectural transformation has taken place, analysis scenarios may be formulated. Analysis scenarios are a simple yet effective way to represent quality concerns as goals to be addressed and achieved during the process of architectural analysis. Quality concerns relate to non-functional requirements (NFRs). They reflect concerns such as system dependability, project cost, schedule and effort, and component concerns such as availability, certification, support and compatibility.

CSAFE allows system designers to create scenarios to perform structural, conformance and quality checks as well as "what-if" analysis. For the purpose of this analysis we have formulated two different scenarios. The first scenario is formulated from the *Requirement Viewpoints* to improve the maintainability, performance and reliability (i.e. availability) of EDDIS. The second scenario is formulated from the Programmer viewpoint, who is interested to improve only performance of EDDIS. The analysis scenarios may be selective (i.e. component or service level) or global (system level). Table 5.3 and Table 5.4 show the concerns identified for the scenarios. Scope identifies aspects of the system affected by a particular concern. Detailed descriptions of these quality concerns are provided in Appendix G.

Table 5.3 EDDIS Scenario descriptions – Scenario 1

Concern	Sub-concern	Description (Refinement)	Scope	Wt.
Component	Availability	Component(Availability) equals to web service	accessLocate	High
Component	Certification	Component(Certification) equals to yes	accessLocate	Medium
Component	Cost	Component(Cost) less than to 500	accessLocate	Low
Component	Standard	Component(Standard) equals to Z39.50	accessLocate	High
Reliability	Availability	Reliability(Availability) greater than or equals to 60	accessLocate	High
Component	Availability	Component(Availability) equals to web service	accessOrder	High
Component	Certification	Component(Certification) equals to yes	accessOrder	Medium
Component	Cost	Component(Cost) less than to 650 yearly	accessOrder	Medium
Component	Standard	Component(Standard) equals to Z39.50	accessOrder	High
Maintainability	Time	Maintainability(Time) less than or equals to 18 months	accessOrder	High
Reliability	Availability	Reliability(Availability) greater than or equals to 65%	accessOrder	Medium
Component	Availability	Component(Availability) equals to web service	accessSearch	High
Component	Certification	Component(Certification) equals to yes	accessSearch	Medium
Component	Cost	Component(Cost) less than to 500	accessSearch	Low
Component	Standard	Component(Standard) equals to Z39.50	accessSearch	High

Concern	Sub-concern	Description (Refinement)	Scope	Wt.
Reliability	Availability	Reliability(Availability) greater than or equals to 60	accessSearch	High
Component	Availability	Component(Availability) equals to inhouse	admin_ services	Low
Component	Version	Component(Version) greater than or equals to 0.3	admin_ services	Low
Maintainability	Requirement	Maintainability(Requirement) equals to user	admin_ services	Low
Component	Availability	Component(Availability) equals to inhouse	document_ services	Low
Component	In	Component(In) less than or equals to 5	document_ services	Medium
Component	Standard	Component(Standard) equals to Z39.50	document_ services	High
Maintainability	Requirement	Maintainability(Requirement) equals to user	document_ services	Low
Maintainability	Time	Maintainability(Time) less than or equals to 18 months	document_ services	Low
Component	Availability	Component(Availability equals to inhouse	user_ validation	Medium
Component	Certification	Component(Certification) equals to yes	user_ validation	High
Component	Version	Component(Version) equals to 4.0	user_ validation	Medium
Maintainability	Technology	Maintainability(Technology) equals to updated	user_ validation	Medium
Maintainability	Time	Maintainability(Time) less than or equals to 12 months	user_ validation	Medium
Business	Cost	Business(Cost) equals to strict	System	Medium
Business	Platform	Business(Platform) equals to Windows 2000/XP	System	High
Business	Schedule	Business(Schedule) equals to strict	System	Medium
Performance	Response Time_PL	Performance(ResponseTime_UPL) less than or equals to 0.75 seconds	System	High
Performance	Response Time_UPL	Performance(ResponseTime_PL) less than or equals to 4 seconds	System	High
Performance	Throughput_ PL	Performance(Throughput_PL) greater than or equals to 150 transaction/per second	System	Medium

Fig. 5.8 shows the scenario derived for the programmer viewpoint (i.e. Scenario 2).

Fig. 5.8 Creating a new analysis scenario 'Scenario 2'

Table 5.4 EDDIS Scenario descriptions – Scenario 2

Concern	Sub-concern	Description (Refinement)	Scope	Wt.
Performance	Response Time_PL	Performance(ResponseTime_UPL) less than or equals to 0.75 seconds	System	High
Performance	Response Time_UPL	Performance(ResponseTime_PL) less than or equals to 4 seconds	System	High
Performance	Throughput_ PL	Performance(Throughput_PL) greater than or equals to 150 transaction/per second	System	High
Component	Availability	Component(Availability) equals to web service	accessLocate	High
Component	Certification	Component(Certification) equals to yes	accessLocate	Medium
Component	Cost	Component(Cost) less than to 500	accessLocate	Low
Component	Standard	Component(Standard) equals to Z39.50	accessLocate	High
Component	Availability	Component(Availability) equals to web service	accessOrder	High
Component	Certification	Component(Certification) equals to yes	accessOrder	Medium
Component	Cost	Component(Cost) less than to 650 yearly	accessOrder	Medium
Component	Standard	Component(Standard) equals to Z39.50	accessOrder	High
Component	Availability	Component(Availability) equals to web service	accessSearch	High
Component	Cost	Component(Cost) less than to 500	accessSearch	Low
Component	Standard	Component(Standard) equals to Z39.50	accessSearch	High
Component	Availability	Component(Availability) equals to inhouse	document_ services	Low

Fig. 5.9 show part of the analysis scenario for the *document_services* service in Scenario 1. The constraints associated with the service and their weightings are shown in the right pane. The bottom left form in the right pane provides a refinement of the selected constraint. The scale information shows indicates lowest and highest possible weighting value for the constraint. The tool uses this information to generate query statements that are used by the mapping processes to locate matching architectural and component solutions.

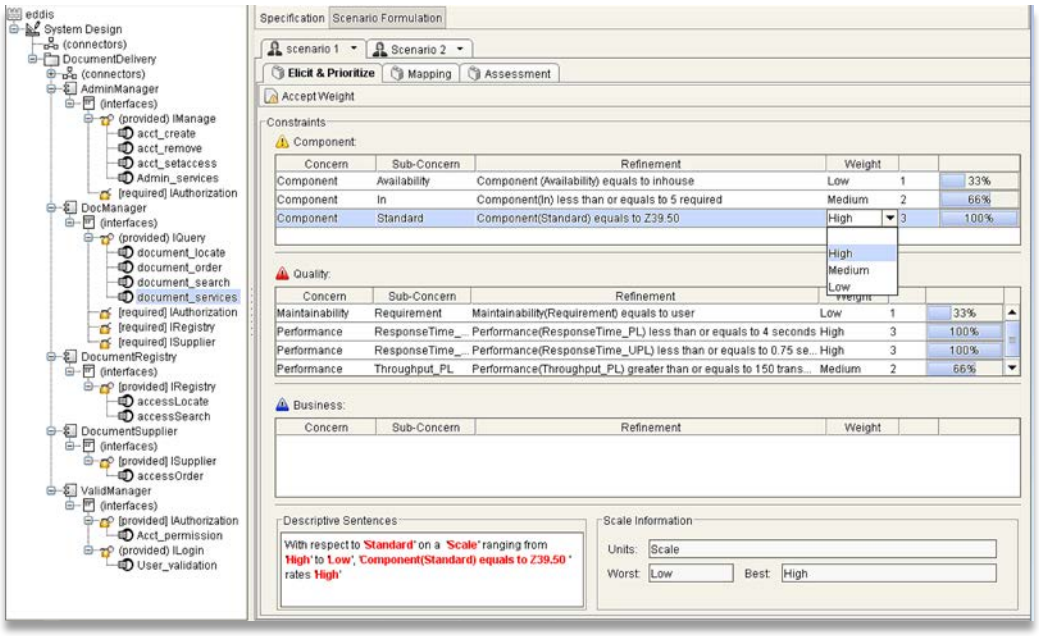


Fig. 5.9 Formulating scenario for *document_services* – Scenario 1

5.3.2 Analysing EDDIS Architecture

The analysis begins with the mapping of a scenario onto architectural design templates as described in Section 4.1.4 (see Fig. 5.10 and Fig. 5.11 for Scenario 1 and Scenario 2 respectively). The output of the scenario mapping process is a set of architectural design templates that best match the qualities and the quality thresholds identified in the analysis scenario. Architectural design templates include design patterns, architectural style and local organisation-defined design schemes. The flexibility

provided by CSAFE means that organisations can add their own custom design templates, design patterns and heuristics.

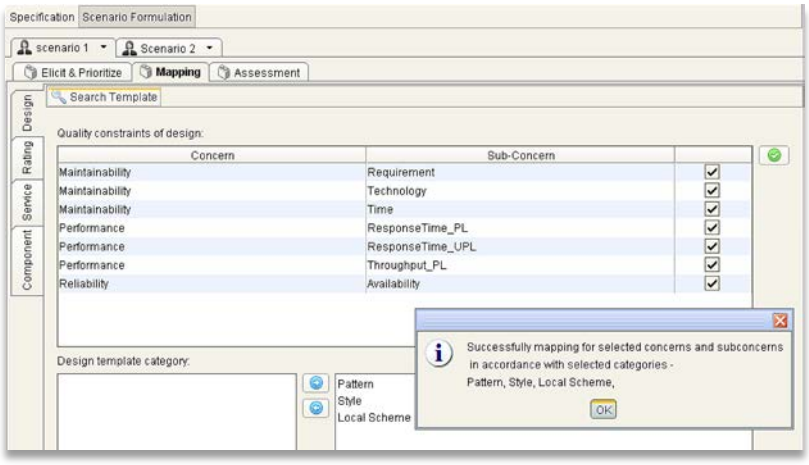


Fig. 5.10 Mapping EDDIS formulated scenarios of Scenario 1 onto Design Template Library

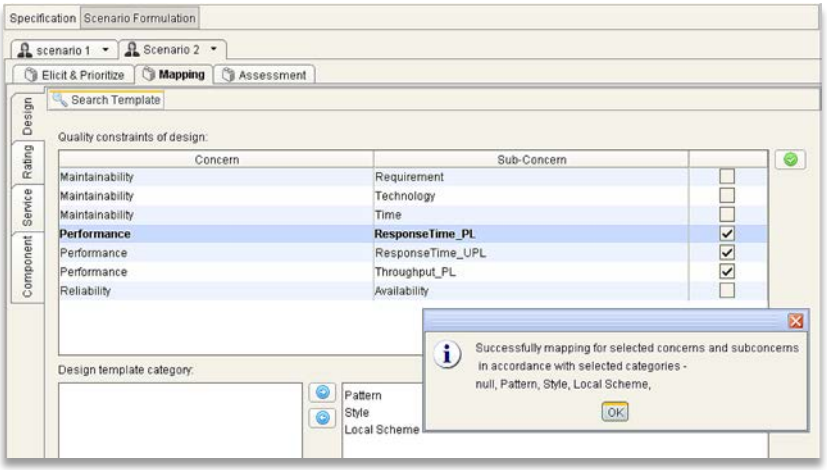


Fig. 5.11 Mapping EDDIS formulated scenarios of Scenario 2 onto Design Template Library

Table 5.5 shows a typical in-house design template called *ServiceOrder Provision*.

Table 5.5 *ServiceOrder Provision* template

Category	Local scheme
Name	*ServiceOrder Provision*
Also-Known-As	Order Provision
Related-Rules	-
Intent	A document may require service of search, locate and order. There is a need to restrict the order service to reside in a component, which consists service search and locate. The program's requirements imply constraints on

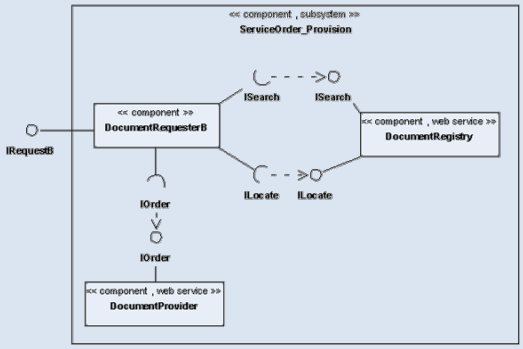
	the order in which threads should access the resources.
Context	When document manager require search, locate and order services, restricted document order in a separate execution is good a strategy. Suppose you are designing an application to manage a document for an online digital library. A component obtain document and location identifiers from an centralize document registry before placing a document order. Document orders are placed with the document supplier component.
Motivation	DocumentManager may require services of DocumentServer which consists of ISearch and ILocate, and DocumentServer which consists of IOrder.
Configuration	
Consequences	Performance.ResponseTime = {the rules provides a way to control, 2} Performance.Throughput = {the rules provides a way to control, 2 Maintainability.Time = {provided a systematic allocation towards maintenance time for the document main services, 3} Maintainability.Requirement = {allows the document server maintain the order service more effectively, 3} Reliability.Availability = {the rule provides a better way to control the availability of related services. Which allow longer duration of order service to be served, 3}

Fig. 5.12 shows three recommended architectural solutions generated as a result of the concerns identified in *Analysis Scenario 1*. The suggested architectural solutions are *Cluster-Server* pattern, *Three-tier proxy server* architectural style and *ServiceOrder provision* architectural style. The recommendations are described in detail in Appendix F. The analysis process rates the architectural design templates based on how well they contribute or lend themselves to critical quality concerns identified in the analysis scenario. When the design templates are rated, they are moved to a solution state where they are instantiated to define the particular variation in the context of EDDIS solutions. The designer notes are entered with the solutions to rationalise the design decisions taken by the system designer. In the case where a recommended design does not contribute to a quality concern a "Not Applicable (N/A)" remark is entered.

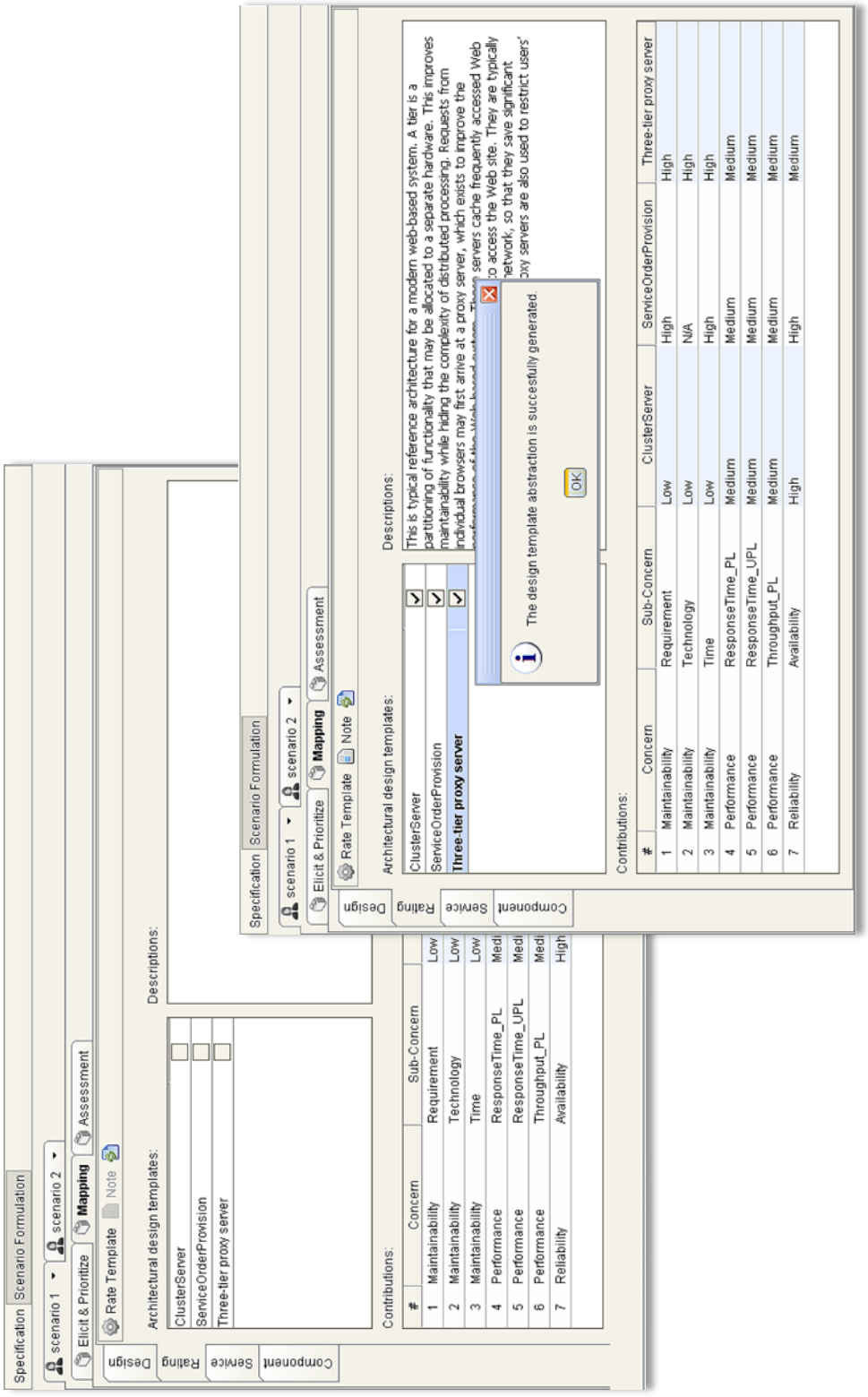


Fig. 5.12 Recommended solutions – Scenario 1

Fig. 5.13 shows the *ClusterServer* pattern dependency and contributions that the template may posses with its scoring values. The figure also shows the design template configuration and XML specification in the design template library.

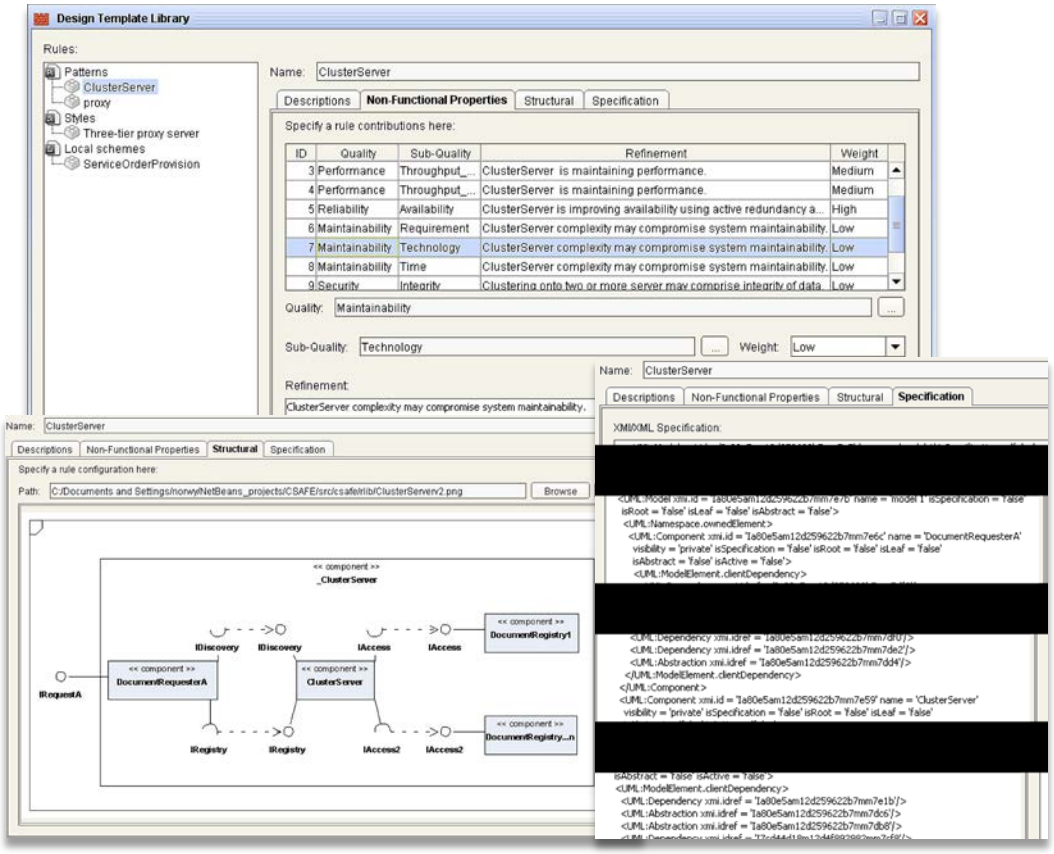


Fig. 5.13 *ClusterServer* pattern with its contributions[2], configuration and specification

The rationale for each recommended architectural solution is provided below:

- *Service-Order Provision.* This architectural style represents a local (in-house) design solution for an online digital library that may require document search, locate and order services. The architectural style enforces the separation of search and locate services, which reside in the same component, from the order service. This may imply that there are constraints on the order in which threads access the

[2] The toolset allows the system designer to record a list of concern/sub-concern and retrieve back thru button click, detail descriptions is described in Appendix F.

resources. However, separation of order services may slightly affect performance of the application response time and throughput. The design improves maintainability by providing a systematic allocation towards maintenance time for the main services, and allowing the document server to maintain the order service more effectively. This architectural style improves system availability by controlling the provision of the order service.

- *Cluster-Server pattern* [Msdn10]. This design enables the system to maintain good performance while improving availability by using active redundancy and automatic restart during failover. However, cluster-server complexity is likely to compromise system maintainability.
- *Three-tier proxy server architectural style* [Bass05]. This is typical reference architecture for a modern web-based system. A tier is a partitioning of functionality that may be allocated to a separate hardware. This improves maintainability while hiding the complexity of distributed processing. Requests from individual browsers may first arrive at a proxy server, which exists to improve the performance of the Web-based system. These servers cache frequently accessed Web pages that users may retrieve without having to access the Web site. They are typically located close to the users often on the same network, so that they save significant communication and computation resources. Proxy servers are also used to restrict users' access to certain Web sites.

Table 5.6 shows how the three alternative designs contribute to the critical quality concerns.

Table 5.6 Architectural design alternatives contributions – Scenario 1

Concerns	Sub-Concerns	Architectural Design Alternatives		
		CS	SOP	TPS
Performance	Response time	Medium	Medium	Medium
	Throughput	Medium	Medium	Medium
Reliability	Availability	High	High	Medium
Maintainability	Requirement	Low	High	High
	Technology	Low	N/A	High
	Time	Low	High	High

Legends: CS – *ClusterServer* SOP – *ServiceOrder Provision* TPS – *Three-tier proxy server*

5.3.3 Revising EDDIS Architecture

The final step involves modifying the EDDIS system architecture to reflect the recommended architectural solutions. The modified architectures will then be rated for their relative contributions to the quality concerns. Fig. 5.14(i) to Fig. 5.14(iii) show the separate EDDIS architectures based on the three design templates. The modification to the original architecture is shown in the boxed area. The mapping process is explained next.

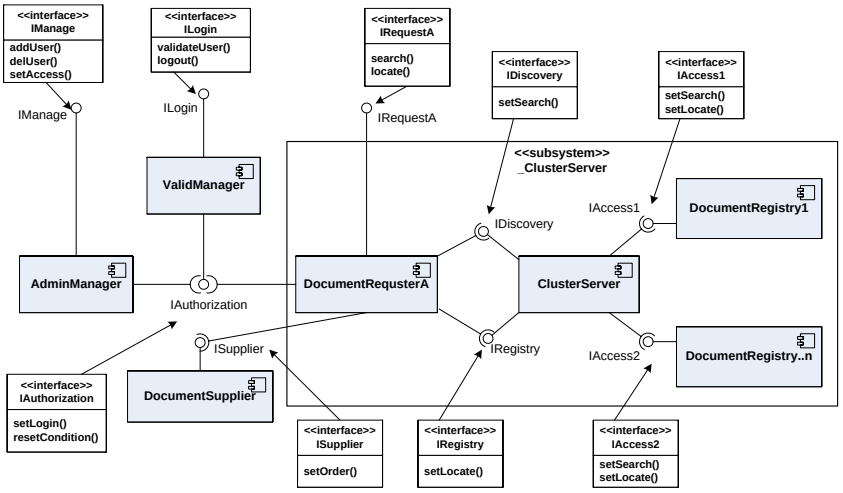


Fig. 5.14(i) *ClusterServer* pattern (S1)

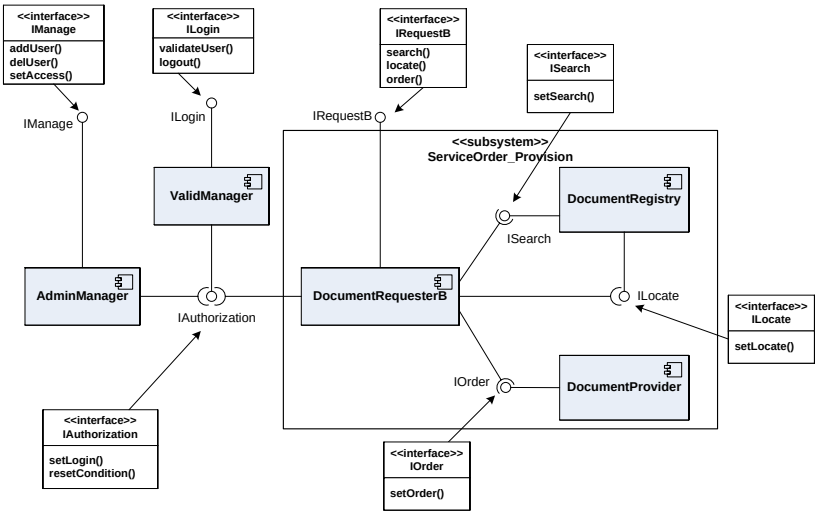


Fig. 5.14(ii) *Service-Order Provision* local-scheme (S2)

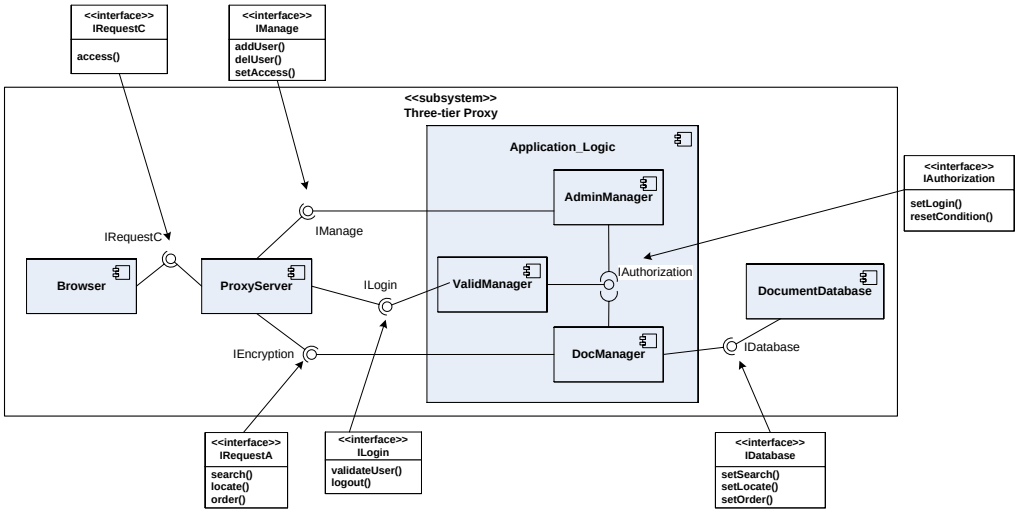


Fig. 5.14(iii) *Three-tier proxy server* architectural style (S3)

CSAFE assists in modifying the initial architecture into the recommended solutions using a two-step process:

- Firstly, it maps the existing system architecture services onto the design template's abstract components. This is done taking into account any specified constraints and design heuristics.
- Secondly, it maps the abstract components onto suitable and available concrete components.

Fig. 5.15 shows how the CSAFE toolset aids in the mapping of EDDIS architecture services onto the abstract components of the design templates. In this example, *document_services* service is mapped onto the *DocumentRequesterB* component of the *ServiceOrder Provisioning* design template. Clicking on the abstract component returns a list of services from which *document_services* is selected. When the mapping is complete, the abstract component *DocumentRequesterB* is associated with four services including *document_locate*, *document order*, *document_search* and *document_services*. The toolset provides a visualisation of these associations (refers to Fig. 5.15).

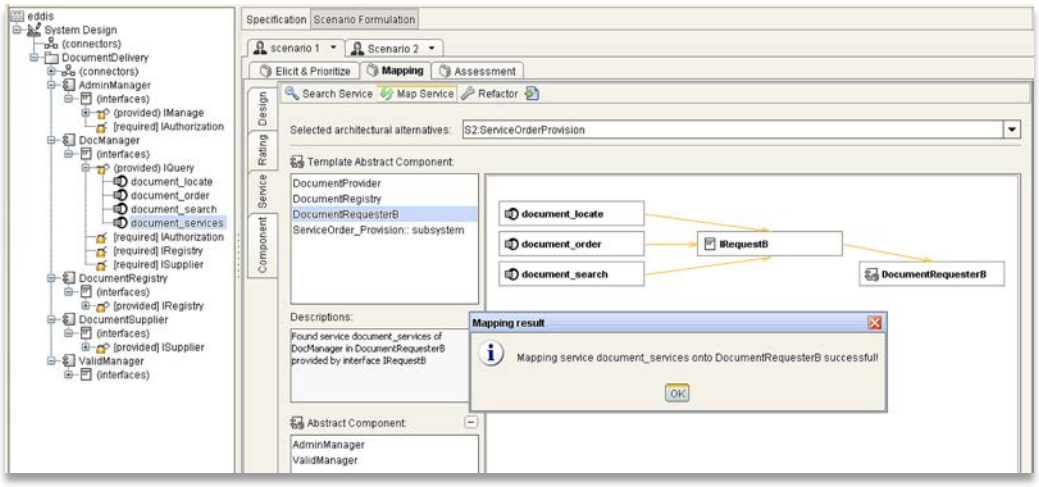


Fig. 5.15 Mapping *document_service* onto *DocumentRequestB* abstract component

If a desired service is not found in the design template, service mapping can still be performed through a refactoring facility provided by tool. Refactoring allows manual mapping and component reconfiguration. Fig. 5.16 shows an example of refactoring that reconfigures the *ValidManager* component in the initial EDDIS architecture for the *ServiceOrder Provision* design template. Fig. 5.17 shows an association diagram of the *AdminManager* and its services after being reconfigured for the *ServiceOrder Provision* design template using refactoring.

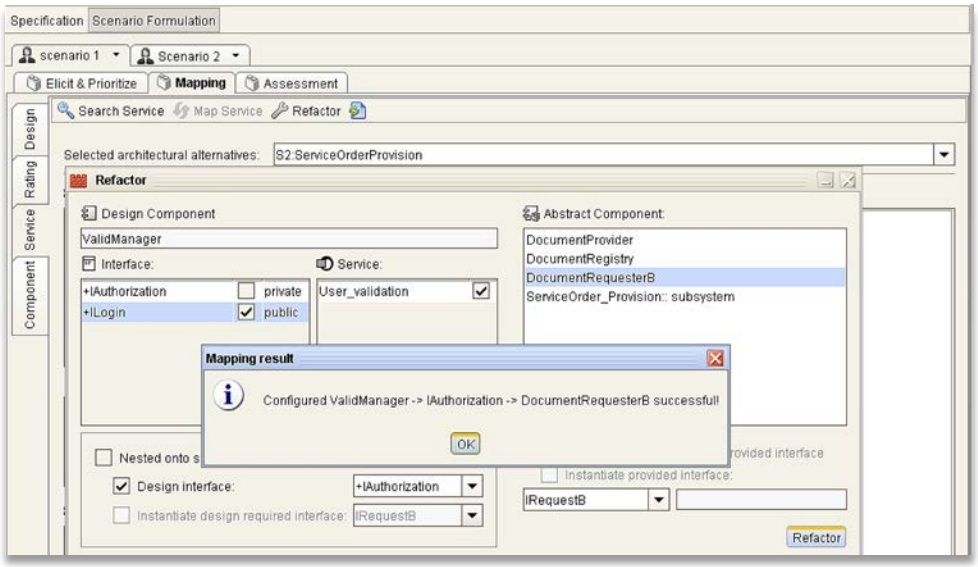


Fig. 5.16 Refactoring *ValidManager* onto the *ServiceOrderProvision*

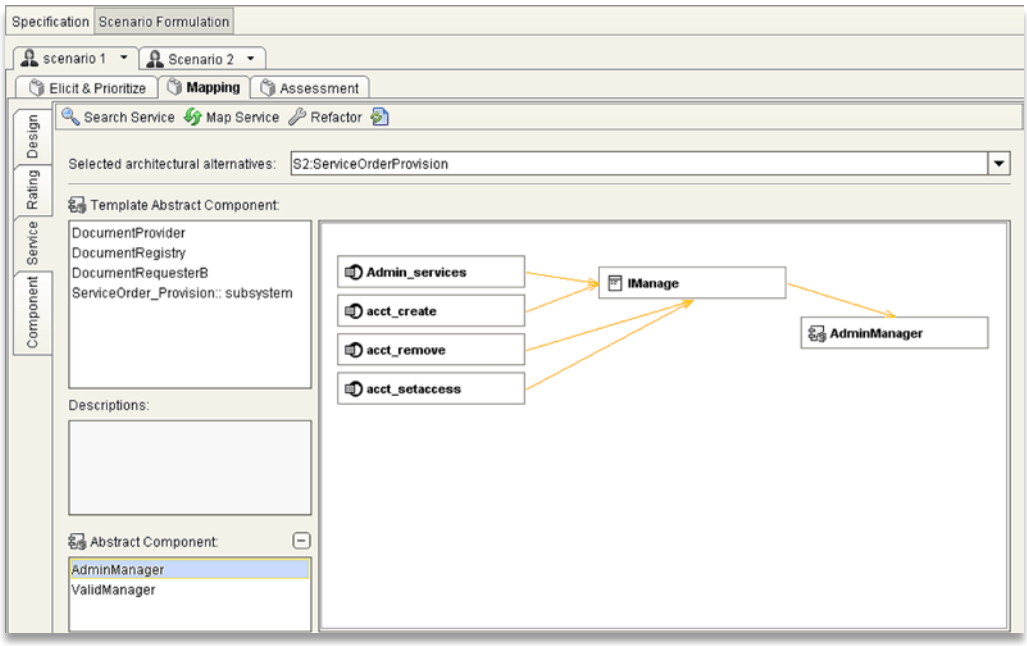


Fig. 5.17 *AdminManager* abstract component with associated services

The second step composes the abstract EDDIS architectures by mapping their abstract components onto concrete components in the repository. The tool provides a quantitative indication of how well each mapping fits and provides further suggestion for component selection. Mismatches are flagged and indicated in colours that correspond to the severity of the mismatches levels (e.g. low, medium or high warning). The severity is borne by the weight assigned during the formulated analysis scenarios which prioritise the concerns. Fig. 5.18 shows the result of mapping the *AdminManager* abstract component to *AdminManager_3* concrete component which has a 66% match. The fitness percentage is calculated based on number of matches divided by number of the component selected concerns. For example, *AdminManager_3* matches two divided by three concerns of the *AdminManager* abstract component. These represent the required component version, which should be greater or equals to 0.3, and the maintainability concern, which equals to user. *AdminManager_3's* version and maintainability are 0.4 and user. While the component's availability property is specified as COTS instead of inhouse. Detailed specifications of the concrete components are available in Appendix C6.

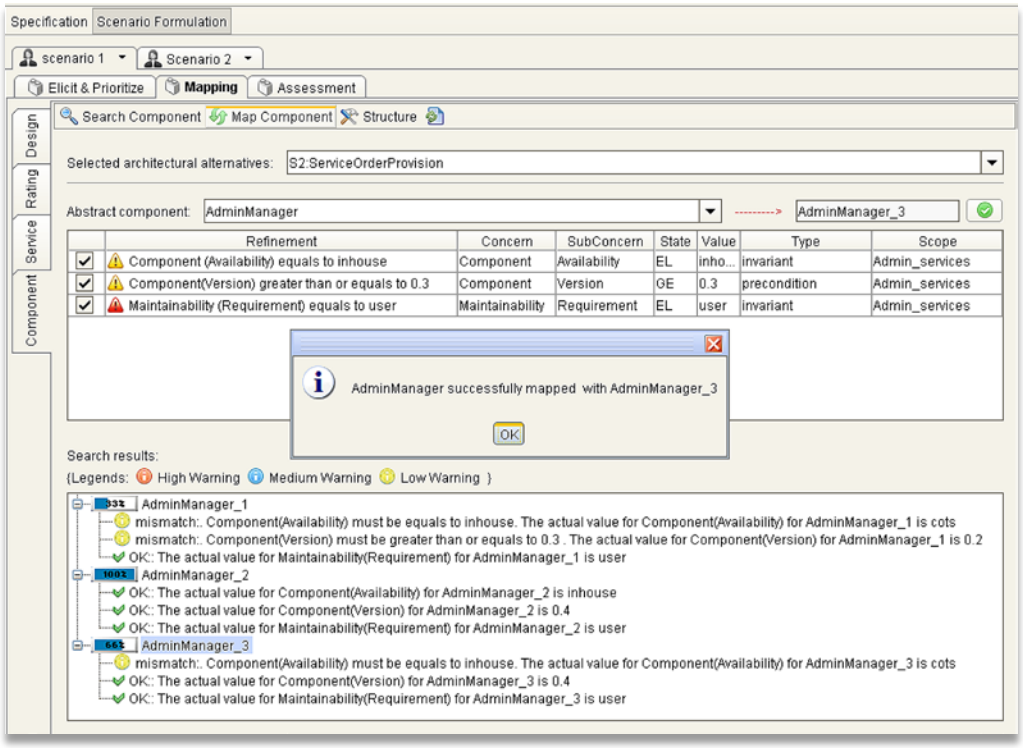


Fig 5.18 Mapping onto concrete component

The structural checker completes the composition process by ensuring structural compatibility between the abstract and the selected concrete component. For example, Fig. 5.19 shows potential mismatches found by the checker between *AdminManager* and AdminManager_3. The checker flagged two error messages: the first indicates the *addUser* signature of *IManage* provided interface has an incompatible method signature, and the second indicates the *deleteUser* signature of *IManage* provided interface has an incompatible parameter type. Nevertheless, the decision is left to the system designer either to proceed with the composition, or to maintain a temporary placeholder for the abstract component until a suitable concrete component is found.

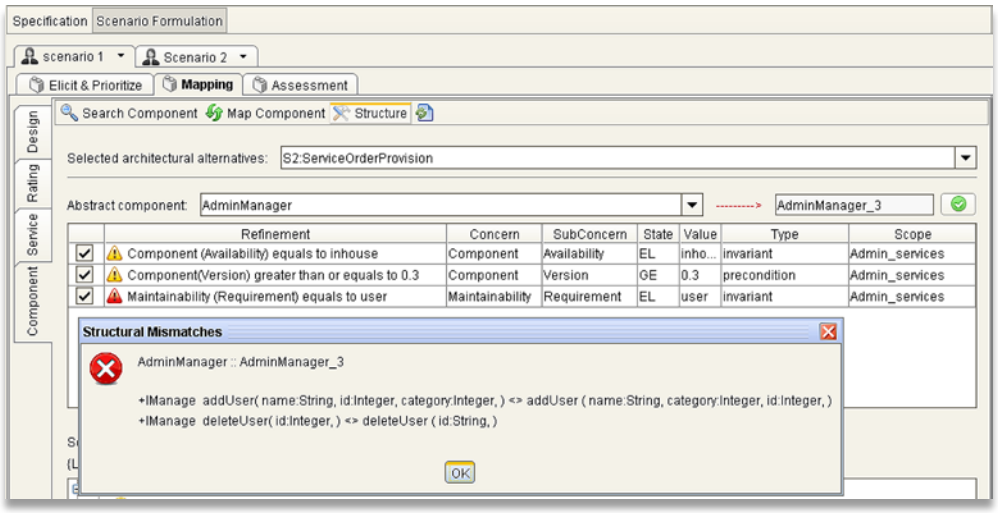


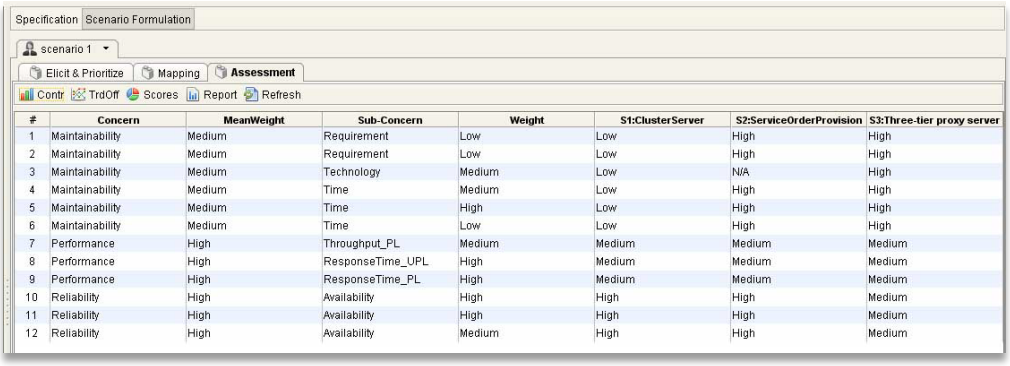
Fig 5.19 Structural mismatch found between *AdminManager* and *AdminManager_3*

Lastly, the design alternatives are assessed by comparing the quality concerns identified in the analysis scenario against the contributions of the design alternatives. Table 5.7 and Fig. 5.20 show the result of the comparison for Scenario 1. While Fig. 5.21 shows the result of the comparison for Scenario 2.

Table 5.7 Comparison of EDDIS concerns and design alternatives contributions – Scenario 1

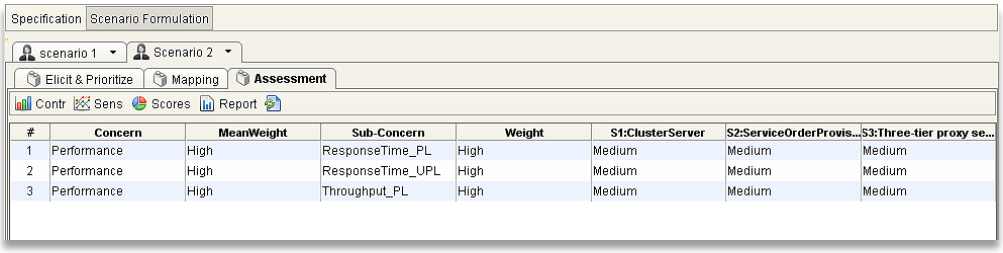
Scenario 1					S1: CS	S2: SOP	S3: TPS
Concern	**Mean Wt.**	**Sub concern**	**Wt.**	**Scope**			
Performance	High	Response time_upl	Medium	S1.2.1	Medium	Medium	Medium
		Response time_pl	Medium	S1.2.1	Medium	Medium	Medium
		Throughput	Medium	S1.2.1	Medium	Medium	Medium
Reliability	High	Availability	High	S1.3.1	High	High	Medium
		Availability	High	S1.4.1	High	High	Medium
		Availability	Medium	S1.5.1	High	High	Medium
Maintainability	Medium	Requirement	Low	S1.2.1	Low	High	High
		Requirement	Low	S2.1.1	Low	High	High
		Technology	Medium	S1.1.1	Low	N/A	High
		Time	Medium	S1.1.1	Low	High	High
		Time	High	S1.5.1	Low	High	High
		Time	Low	S1.2.1	Low	High	High

Legends: CS – *ClusterServer* SOP – *ServiceOrder Provision* TPS – *Three-tier proxy server*



#	Concern	MeanWeight	Sub-Concern	Weight	S1:ClusterServer	S2:ServiceOrderProvision	S3:Three-tier proxy server
1	Maintainability	Medium	Requirement	Low	Low	High	High
2	Maintainability	Medium	Requirement	Low	Low	High	High
3	Maintainability	Medium	Technology	Medium	Low	N/A	High
4	Maintainability	Medium	Time	Medium	Low	High	High
5	Maintainability	Medium	Time	High	Low	High	High
6	Maintainability	Medium	Time	Low	Low	High	High
7	Performance	High	Throughput_PL	Medium	Medium	Medium	Medium
8	Performance	High	ResponseTime_UPL	High	Medium	Medium	Medium
9	Performance	High	ResponseTime_PL	High	Medium	Medium	Medium
10	Reliability	High	Availability	High	High	High	Medium
11	Reliability	High	Availability	High	High	High	Medium
12	Reliability	High	Availability	Medium	High	High	Medium

Fig. 5.20 Assessing quality concerns and architecture design solutions - Scenario 1



#	Concern	MeanWeight	Sub-Concern	Weight	S1:ClusterServer	S2:ServiceOrderProvis...	S3:Three-tier proxy se...
1	Performance	High	ResponseTime_PL	High	Medium	Medium	Medium
2	Performance	High	ResponseTime_UPL	High	Medium	Medium	Medium
3	Performance	High	Throughput_PL	High	Medium	Medium	Medium

Fig. 5.21 Assessing quality concerns and architecture design solutions - Scenario 2

The Fig. 5.22 and Fig. 5.23 show the weighted contributions of the different design alternatives for Scenario 1[3]. S1 offers the poorest solution as it has an overall quality contribution score of 0.641. Of the remaining, S2 has the better score of 0.818 and S3 a slightly lower score of 0.793. Although, S2 looks like the best design, it may not necessarily be chosen. For example, the cost of implementing the system using S2 may be beyond the organisation's budget. To decide on the most acceptable architecture, stakeholders need to explore how each suggested design relates to critical EDDIS sub-concerns (see Fig. 5.24).

[3] Details weighting and scoring values are compiled in Appendix D, Table D5.1

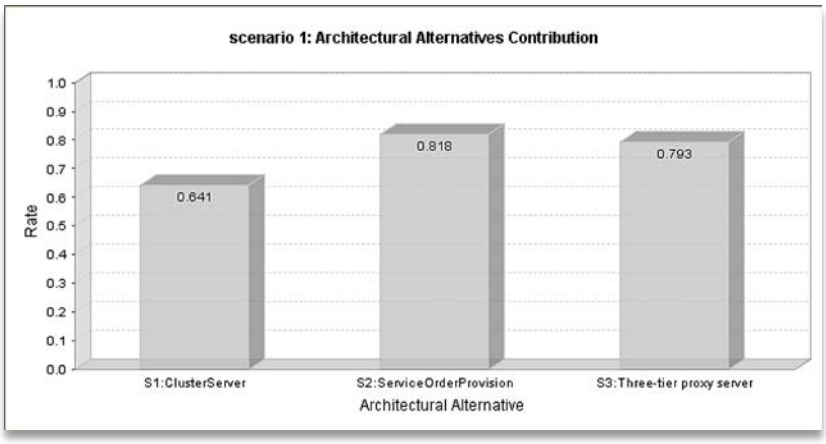


Fig. 5.22 Contribution of suggested alternatives according to overall – Scenario 1

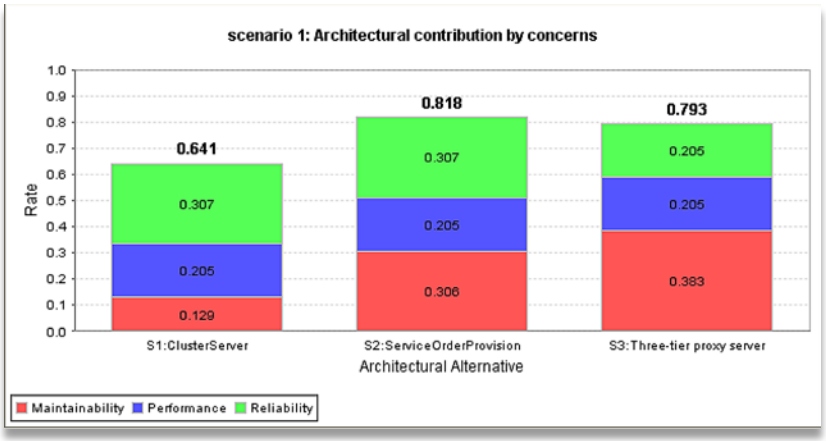


Fig. 5.23 Contribution of suggested alternatives according to main concerns – Scenario 1

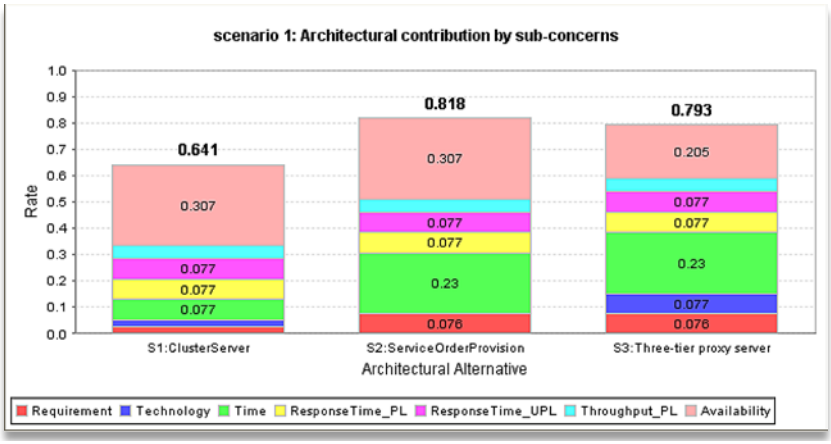


Fig. 5.24 Contribution of suggested alternatives according to sub-concerns – Scenario 1

The Fig. 5.25 shows the weighted contributions of the different design alternatives for Scenario 2[4]. In Scenario 2, all the three design alternatives offer an equivalent quality contribution score of 0.666. The contributions at sub-concern level are also equivalent (see Fig. 5.26). However, the programmer concern for performance is only moderately addressed by all the alternative designs.

In this particular case, the preferred architectural solution is selected from the design alternatives in Scenario 1 as the alternatives offer the same contribution for performance in Scenario 2. However, in cases where design alternatives offer varying contributions for different scenarios, further negotiation (trade-offs) may be required to resolve the competing scenarios and establish an acceptable compromise. This may involve weighting the stakeholder scenarios.

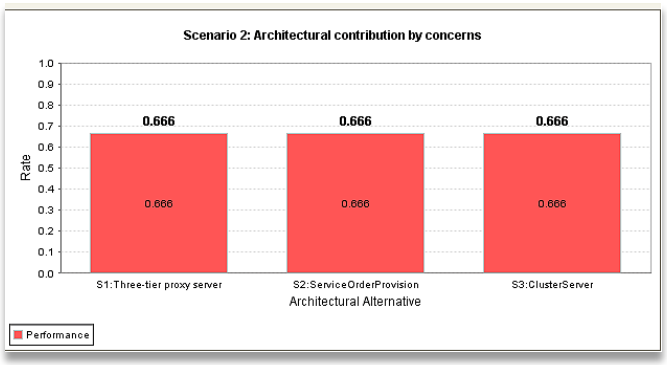


Fig. 5.25 Contribution of suggested alternatives according to performance concern – Scenario 2

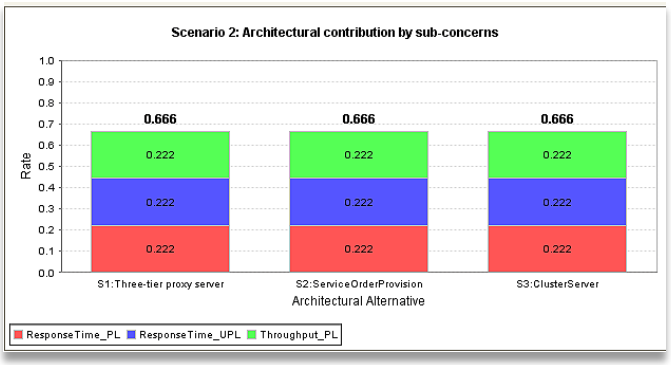


Fig. 5.26 Contribution of suggested alternatives according to performance sub-concerns – Scenario 2

[4] Details weighting and scoring values are compiled in Appendix D, Table D5.2

Sensitivity analysis

As part of the evaluation, two sensitivity analyses were conducted for Scenario 1 to examine how robust the choice of architectural solution was to changes in the relative weightings of critical quality concerns. The first sensitivity analysis examined how the benefit value of the alternative designs might be affected by relative changes in the weight of performance. The second sensitivity analysis examined how value of benefits offered by the alternative designs might be affected by relative changes in the weight of maintainability. Before changes the relative weights for the different quality concerns were: maintainability, 0.38; performance, 0.31; and reliability, 0.31.

Fig. 5.27 shows how the value of benefits for the architectural design alternatives varies with changes in the weight placed on performance. If performance had a weight of zero, the three performance sub-concerns would also have zero weights. After re-normalisation, this would result in weights of 0.44 and 0.56 for reliability and maintainability, respectively. At this point, S2 offers the highest level of benefits followed by S3. S1 has the lowest benefit value. At the other extreme, if performance had a weight of 100 (and therefore maintainability and reliability weights of zero) all the three alternatives designs would have gradual decreasing aggregate benefit values of 0.66. However, since performance has a weight of 0.31, the software designer might consider S2 and S3 marginally attractive solutions.

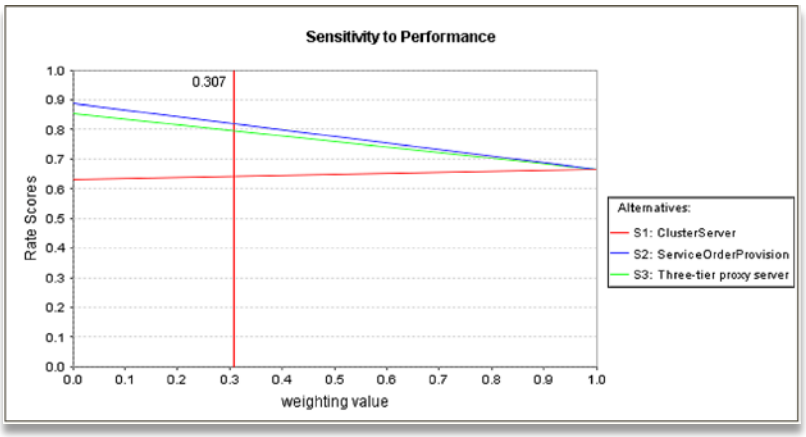


Fig. 5.27 Sensitivity analysis of Performance – Scenario 1

CSAFE also supports sensitivity analysis at sub-concern level. The sensitivity analysis graph for the throughput, a sub-concern of performance, is shown in Fig. 5.28. The design alternatives, S2 and S3, become more attractive when throughput is assigned a weight of 0.07. The system designer would need to conduct similar analysis for other sub-concerns of performance to complete the verification.

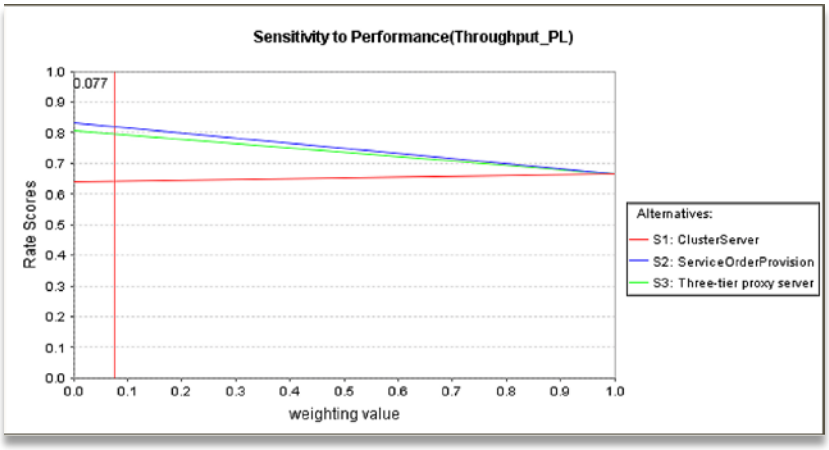


Fig. 5.28 Sensitivity analysis of Performance(Throughput) – Scenario 1

The second sensitivity analysis examines focuses on maintainability. Fig. 5.29 shows how the value of benefits for the architectural design alternatives varies with changes in the weight placed on maintainability. If maintainability had a weight of zero, this would imply that the six maintainability sub-concerns would also have zero weights. After re-normalisation, this would leave weights of 0.50 and 0.50 for performance and reliability, respectively. This would mean, for example, that S1 and S2 would have an aggregate benefit value of 0.833.

At the other extreme, if maintainability had a weight of 100 (and therefore performance and reliability a weight of zero) S3 would have an aggregate benefit value of 1.0. The line joining these points shows the value of benefits for S2, for maintainability weights between 0 and 100. As can be seen, S2 has the highest value of benefits as long as the weight placed on maintainability is less than 0.44. If the weight is above this level then S3 has the highest level of benefits. However, since a weight of

0.36 was assigned to maintainability, it would take a fairly moderate change in this weight before S3 was worth considering. No changes in the weighting attached to maintainability would make the other design alternatives achieve the highest value of benefits, and the software designer can be reasonably confident about selecting S2.

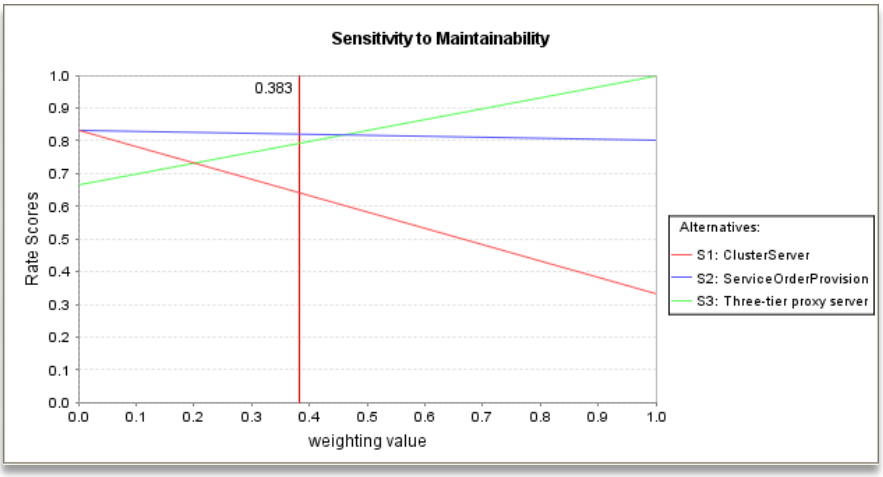


Fig. 5.29 Sensitivity analysis of Maintainability – Scenario 1

Fig. 5.30 shows the sensitivity analysis graph for the EDDIS requirement concern, which is a sub-concern of maintainability. Again the suggested alternatives, S2 and S3, have very close scores for weights between 0 and 100.

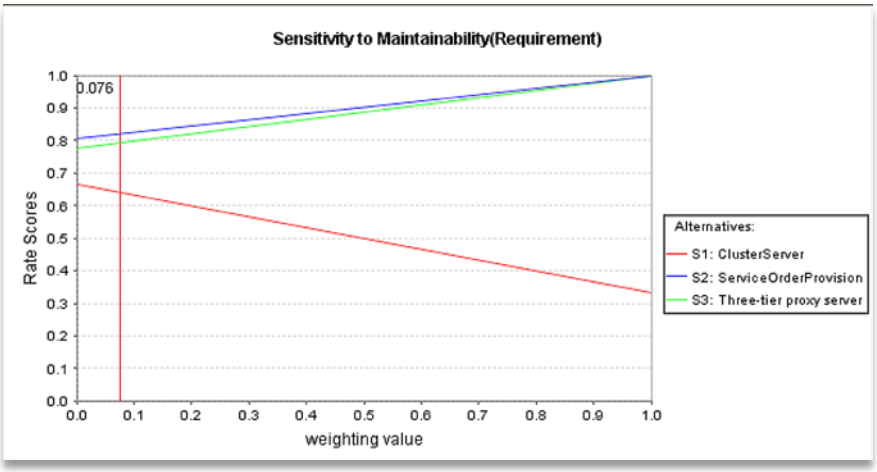


Fig. 5.30 Sensitivity analysis of Maintainability(Requirement) – Scenario 1

Another useful comparative analysis tool provided by CSAFE is the ability to compare weighted quality concerns (i.e. as identified in analysis scenarios) with rating values of suggested architectural design alternatives. The comparison charts are shown in Fig. 5.31 –5.33. The left column shows the scenario ratings of quality concerns, and the right column the contribution ratings of the *ClusterServer* pattern, *ServiceOrder Provision* and *Three-tier proxy server* architectural styles.

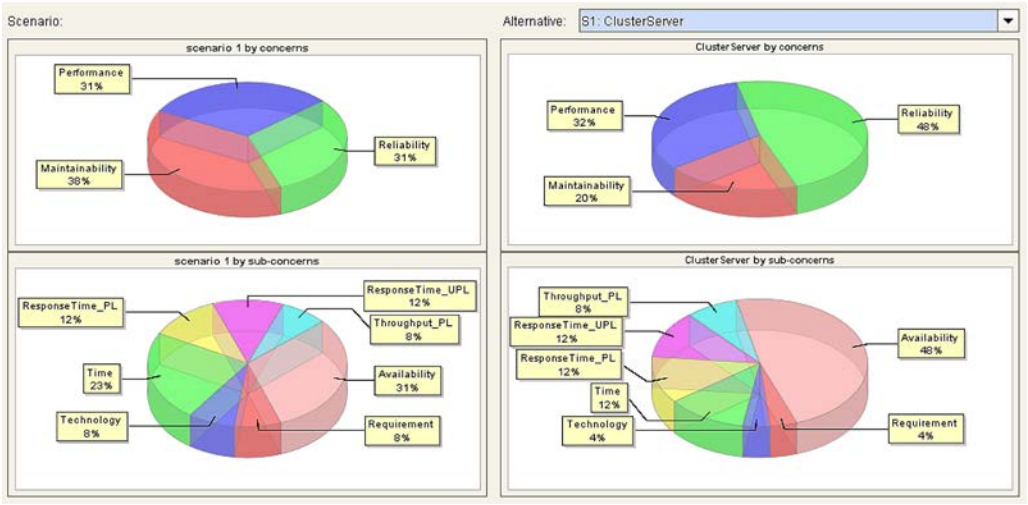


Fig. 5.31 Scoring percentage of *ClusterServer* pattern – Scenario 1

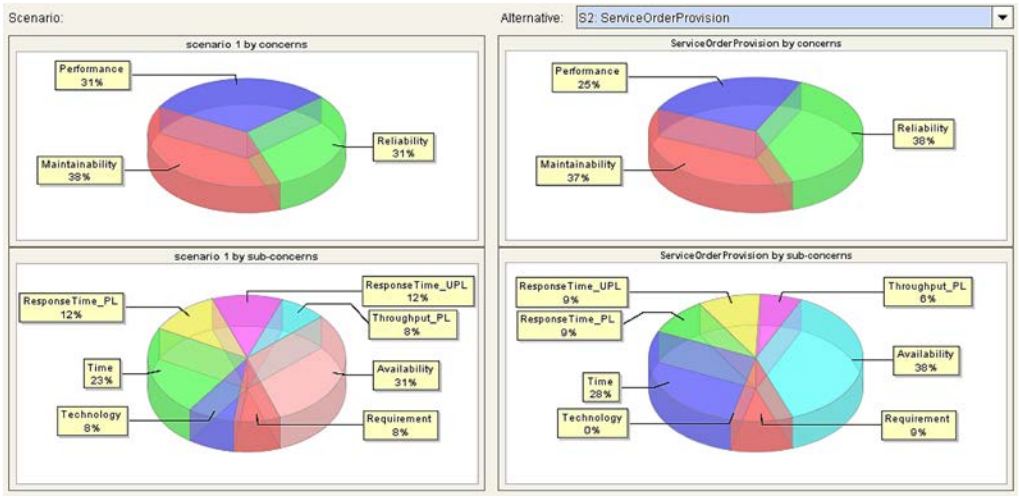


Fig. 5.32 Scoring percentage of *ServiceOrder Provision* local-scheme – Scenario 1

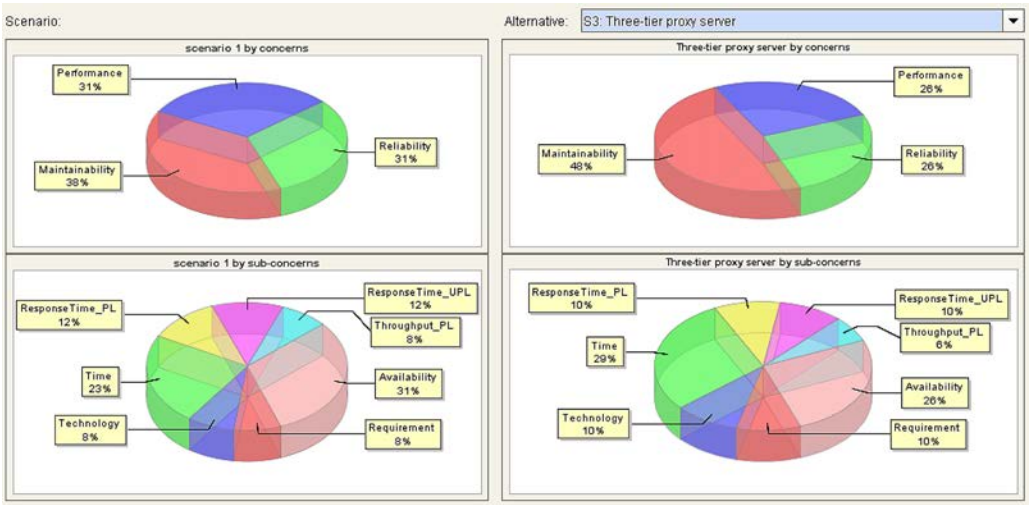


Fig. 5.33 Scoring percentage of *Three-tier proxy server* architectural style – Scenario 1

The detailed report of the results and analysis process generated by the CSAFE tool is available in Appendix D6.

5.4 Summary

This chapter demonstrated the key features and practicability of CSAFE using a subset of requirements extracted from the specification of a real software project, EDDIS. In addition, the evaluation was conducted in the context of two different stakeholder analysis scenarios to demonstrate CSAFE's support for broad stakeholder involvement in architectural design and analysis.

The evaluation started with the description of the case study. This was followed by the construction of the baseline architecture for EDDIS. The baseline architecture was then analysed according to the to the steps in the CSAFE approach. The analysis begun with transformation of EDDIS architectural design to iXML ADL followed by the formulation of two analysis scenarios. The analysis scenarios were used to generate design templates that were in turn used to revise the baseline EDDIS architecture. Lastly, the alternative designs were mapped concrete components and assessed for contributions to the quality concerns identified in the analysis scenarios.

Chapter 6

Evaluation 2: Guided Vehicle Parking Systems

This chapter describes second evaluation of CSAFE. The first evaluation provided a practical demonstration of the CSAFE features discussed in chapter 4, and showed how the approach could be used to improve the quality of software architecture through a process of analysis and refinement. However, the assessment of architectural refinements in the first evaluation was based solely on static analysis. The assessment the refinements relied largely on the documented relationships between design templates and system quality properties. The second evaluation focuses on runtime evaluation to validate architectural refinements. The evaluation assesses the effect of architectural refinements by comparing the runtime behaviour of an existing system against its refined version. The architectural refinements evaluated in the case study are intended to improve the system efficiency and performance. The case study used in this evaluation is derived from an undergraduate software engineering project run at Lancaster University for computer science students. The project is organised around a group of 4-5 students and runs for 25 weeks. The aim of the project is to develop a

simulated *Guided Vehicle Parking System (GVPS)* to provide drivers entering the university campus with accurate and timely information on parking. The case study uses the results from the best GVPS project of the year 2006/2007.

The evaluation starts with the description of the GVPS case study. This is followed by a summarised discussion of the architectural analysis performed on the GVPS and a discussion of the architectural solution adopted. The evaluation concludes with a discussion of three experiments conducted to gauge the effectiveness of the CSAFE refinements on the GVPS runtime architecture.

6.1 The Case Study

The GVPS consists of two main sub-systems: an In-Vehicle Display (IVD) and a Control Centre sub-system. The IVD allows drivers entering the university campus to be assigned the best available parking space, closest to their destination. Drivers select their destination on the IVD as they enter the campus. The IVD communicates with a central server to display a map of campus roads and car parks, highlighting the route to be taken to the selected destination. The IVD also indicates the correct direction to be taken at junctions and roundabouts, both visually and audibly. The IVD informs drivers of road closures and indicates alternative routes when appropriate. When leaving the university, the IVD provides directions back to the exit. The Control Centre sub-system is used by GVPS system administrators to register vehicles, to monitor the status of vehicles and car parks, and to close and open sections of road for emergency or maintenance.

Fig. 6.1 shows the use-cases associated with high-level service descriptions of the GVPS functionality. In the system design these represent services that are later partitioned into abstract, design-time components. The use cases have been extracted from the student project document [Summers06]. Detailed GVPS requirements are provided in Appendix E1.

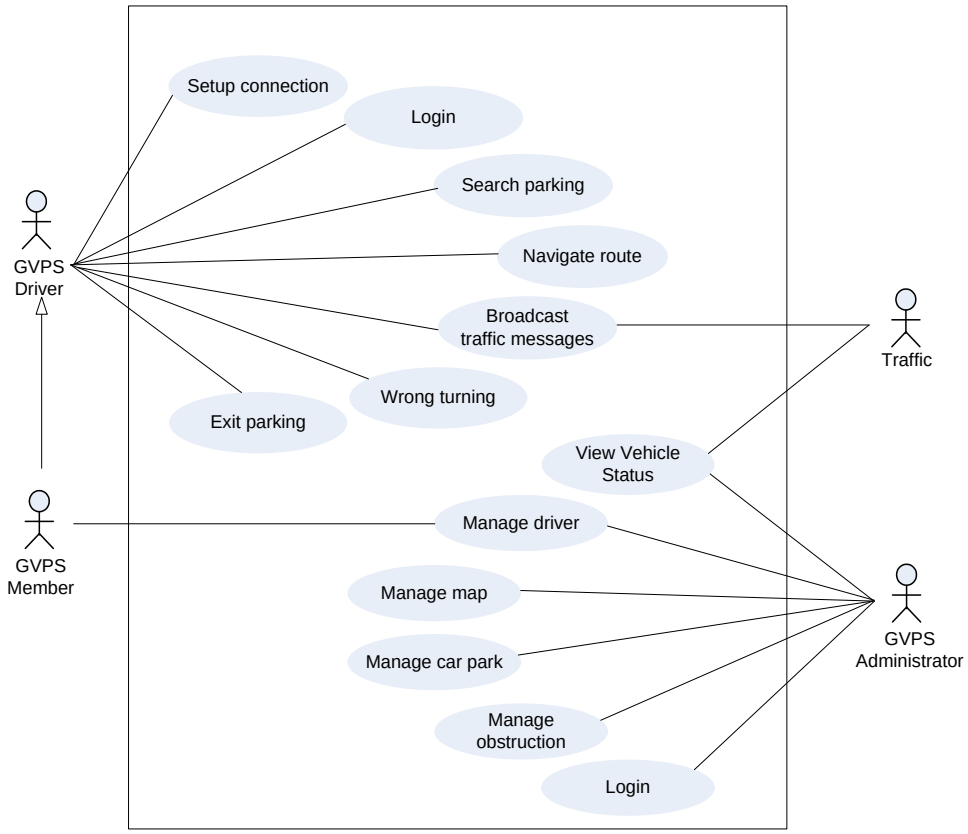


Fig. 6.1 GVPS use-case diagram

6.2 GVPS Viewpoints and Requirements

The viewpoint approach described in Chapter 4 was used to structure and partition the GVPS requirements. Table 6.1 identifies the viewpoints associated with the GVPS and their requirements. The GVPS requirements are derived from four viewpoints: driver (Vp_1), administrator (Vp_2), traffic (Vp_3) and Consortium (Vp_4). A *driver* is a person who interacts with the In Vehicle Display (IVD). The IVD helps the driver to navigate the campus roads to locate suitable parking and to exit the campus. A *driver* is either member of Lancaster University staff or a visitor. University members are required to register their vehicles with the GVPS management to ensure appropriate parking areas are assigned to them (based on permit type).

An unrecognised vehicle is assumed to be a *driver* of type visitor and is assigned a temporary ID. A temporary ID allows a driver to park in visitor areas only. However, a visitor may indicate a disability requirement, in which case the driver is assigned a disabled visitor space. An *administrator* is a person who is responsible for managing the GVPS system. An *administrator's* responsibilities include: driver registration, vehicle monitoring, parking areas monitoring and road management. An administrator is able to view all the vehicles on campus roads using the GVPS. The *consortium* represents the organisation commissioning the GVPS. The *consortium* includes Lancaster University and the project financiers. *Traffic* represents the traffic sensors on the campus roads (e.g. traffic lights and traffic signs).

Table 6.1 GVPS viewpoints and requirements

Viewpoint		**Requirement**			
ID	**Role/Type**	**ID**	**Description**	**Service**	**Ranking**
Vp_1	GVPS_Driver (Operator)	R1.1	To enable drivers either holding a car permit or visitor to access GVPS 24/7.	S1.1.1	Important
		R1.2	To be able drivers to logon to the system using valid RFID or vehicle registration number.	S1.2.1	Essential
		R1.3	To guide the driver of the vehicle to a designated parking place (given as a particular car park) as close to the destination as possible.	S1.3.1	Essential
		R1.4	The display in the vehicle shall show the position of the vehicle on a map.	S1.4.1	Essential
		R1.5	To guide the driver of the vehicle to an exit.	S1.5.1	Essential
		R1.6	To inform drivers of traffic messages according to driver location and distance to an incident	S1.6.1	Important
		R1.7	To inform drivers of when a wrong turning is made and to re-calculate route	S1.7.1	Essential
Vp_2	GVPS_ Administrator (Operator)	R2.1	To enable the admininsrator to access GVPS 24/7 in a secure way.	S2.1.1	Essential
		R2.2	To manage driver accounts i.e. add/delete/update accounts.	S2.2.1	Essential
		R2.3	To manage road maps i.e. add/delete map.	S2.3.1	Essential
		R2.4	To manage car parks on campus by providing their status.	S2.4.1	Important
		R2.5	To enable closure of sections of road in case of emergency or maintenance.	S2.5.1	Important
		R2.6	To monitor the status of all vehicles accessing GVPS.	S2.6.1	Essential

Viewpoint		Requirement			
ID	**Role/Type**	**ID**	**Description**	**Service**	**Ranking**
Vp_3	Traffic (Component)	R3.1	The IVD client shall act as an observer for traffic signal broadcast.	S3.1.1	Important
Vp_4	GVPS_ Consortium (Organisation)	R4.1	The system shall ensure a reasonable level of performance is maintained across the services at all times.		Essential
		R4.2	The system shall provide 24/7 access.		Useful
		R4.3	The system shall enforce authentication policies to avoid loss of data integrity or confidentiality		Essential
		R4.4	The system shall promote XML data map format and driver independence on map resources.		Important
		R4.5	The system shall be developed according to agreed schedule and cost estimate.		Useful

Fig. 6.2 shows the partitioning of GVPS services derived from viewpoints Vp_1–Vp_4. The constraints are indicated with different colours to distinguish their types.

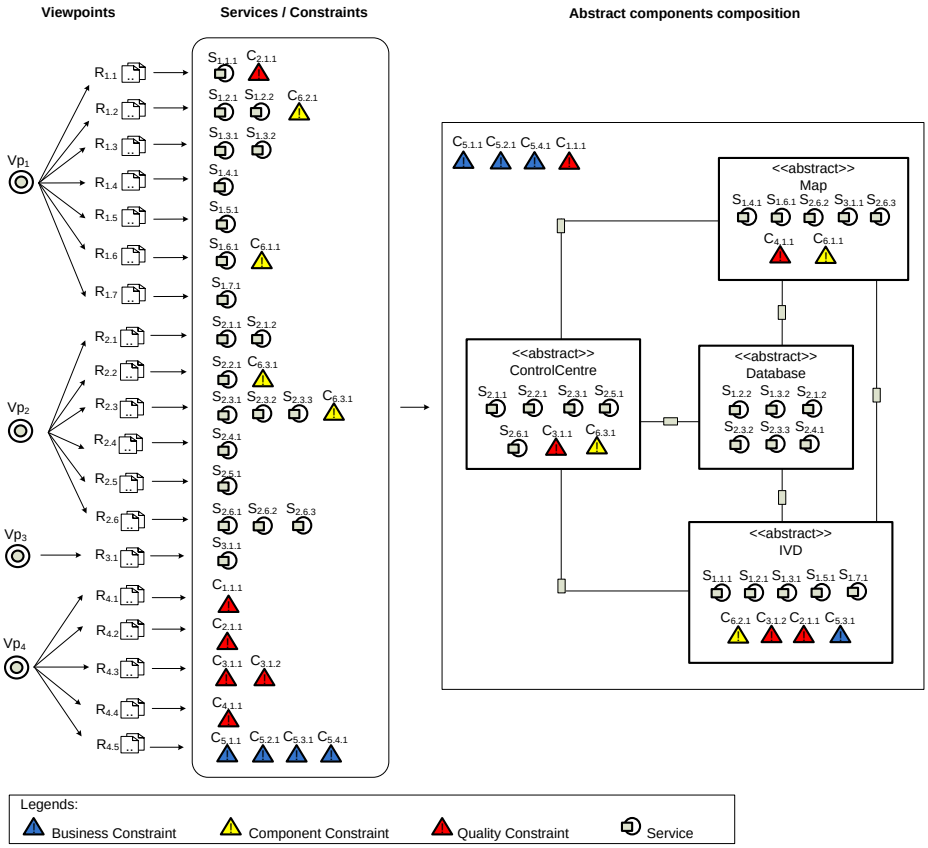


Fig. 6.2 GVPS typical component partitioning

6.3 The Analysis

6.3.1 Documenting the GVPS Architecture

In this case study, GVPS architecture was fully specified using the iXML ADL, Table 6.2 shows a snippet of the iXML description for *CC_Console*. The complete iXML specification is provided in Appendix E2. The GVPS architecture is shown in Fig. 6.3 [Summers06].

Table 6.2 iXML description of *CC_Console*

```
<component name.id = 'CC_Console' type = '' visibility = 'private'>
<component.description>
CC_Console component is for administrative users who can monitor the status of each vehicle and car park on campus, and enable closure of sections of road in case of emergency or maintenance.
</component.description>
<component.interface  name.idref = 'IDataCentre' port.idref = 'r'/>
<component.interface  name.idref = 'IMapCC' port.idref = 'r'/>
<component.interface  name.idref = 'IControlCentre' port.idref = 'p'/>
<component.interface  name.idref = 'IRouteObs' port.idref = 'p'/>
<component.connector name.idref = 'IDataCentre -> CC_Console'/>
<component.connector name.idref = 'IMapCC -> CC_Console'/>
 <component.constraint concern = 'Security' subconcern = 'Integrity' type = 'invariant' state = 'EL' value = ' authentication_policies' scope = 'Login'/>
<component.constraint concern = 'Component' subconcern = 'Persistent' type = 'precondition' state = 'EL' value = 'SQL Server' scope = 'ManageDriver'/>
<component.constraint concern = 'Component' subconcern = 'Persistent' type = 'precondition' state = 'EL' value = 'SQL Server' scope = 'ManageMap'/>
<component.property concern = 'Component' subconcern = 'Availability' value = 'inhouse'/>
</component>
```

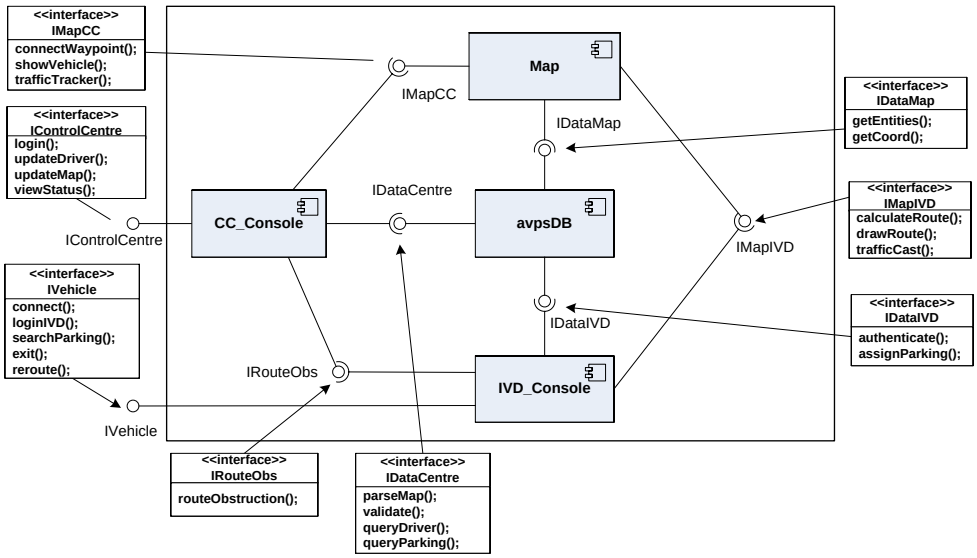


Fig. 6.3 GVPS architectural description with interface identification

The transformed GVPS architecture is shown in Fig. 6.4. The nodes with *gvps* as root represent the composite component, *Navi,* which encapsulates four other components: *CC_Concole*, *IVD_Console*, *Map* and *avpsDB*. The *System Design* node corresponds to the overall GVPS specification. The corresponding component services and interfaces can be seen in Fig. 6.5.

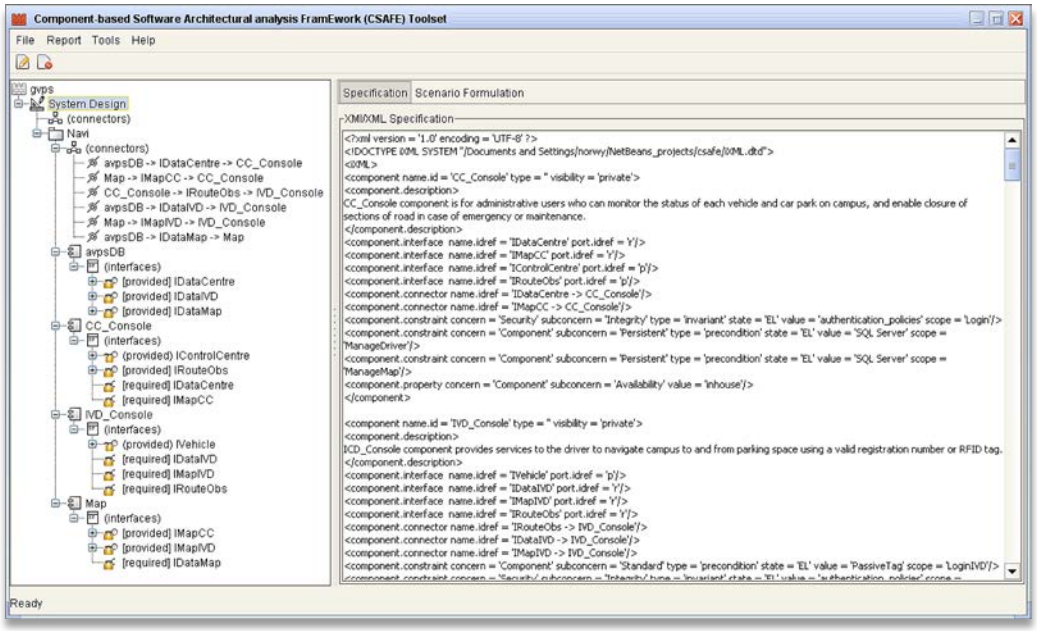


Fig. 6.4. GVPS architecture (left panel) and iXML specification (right panel)

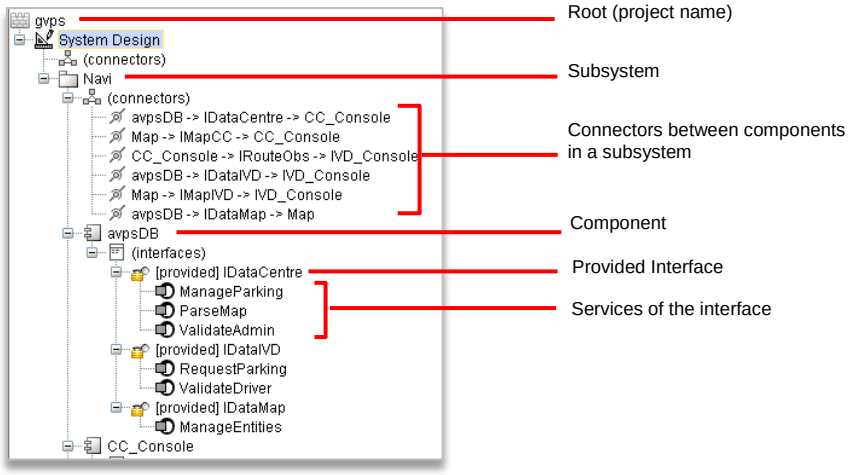


Fig. 6.5 GVPS architecture components and their associated interfaces and connectors

6.3.2 Formulating GVPS Analysis Scenarios

The GVPS requirement specification, GVPS actors and stakeholders discussions were used to elicit and organize the quality concerns for analysis scenarios. The analysis scenario formulated for this evaluation is shown in Table 6.3 (i.e. Scenario 1). The concerns reflect system construction constraints and user expectation of how the system services should be provided. The concerns are weighted to reflect their value in the system from the perspective of requirement viewpoints i.e. interator and non-interator.

Table 6.3 GVPS Scenario descriptions – Scenario 1

Concern	Sub-concern	Description (Refinement)	Wt.	Scope
Flexibility	Expendability	Flexibility (Expendability) equals to xml-based	Medium	DrawMap
Business	Platform	Business(Platform) equals to Windows Mobile	Medium	Exit
Security	Integrity	Security(Integrity) equals to authentication policies	High	Login
Security	Integrity	Security(Integrity) equals to authentication policies	High	LoginIVD
Component	Standard	Component(Standard) equals to PassiveTag	High	LoginIVD
Business	Platform	Business(Platform) equals to Windows Mobile	Medium	LoginIVD

Concern	Sub-concern	Description	Wt.	Scope
Component	Persistent	Component(Persistent) equals to SQL Server	Medium	ManageDriver
Component	Version	Component(Version) greater than to 2.0	High	ManageDriver
Component	Persistent	Component(Persistent) equals to SQL Server	Medium	ManageMap
Business	Platform	Business(Platform) equals to Windows Mobile	Medium	SearchParking
Reliability	Availability	Reliability(Availability) equals to 24/7	Low	SetupConn
Performance	ResponseTime _UPL	Performance(ResponseTime_UPL) less than or equals to 0.5 seconds	High	System
Performance	ResponseTime _PL	Performance(ResponseTime_PL) less than or equals to 4 seconds	High	System
Business	Cost	Business(Cost) equals to moderate	Low	System
Business	Schedule	Business(Schedule) equals to moderate	Low	System
Business	Component Model	Business(ComponentModel) equals to JavaBeans	Medium	System
Component	Availability	Component(Availability) equals to inhouse	Medium	TrafficSignal
Business	Platform	Business(Platform) equals to Windows Mobile	Medium	WrongTurning
Efficiency	Memory	Efficiency (Processor) equals or less than to 20% threshold	High	VehicleTracker
Efficiency	Processor	Efficiency(Memory) equals or less than 75% threshold	High	VehicleTracker

6.3.3 Analysing GVPS Architecture

The GVPS use services that consume significant system resources such route plotting, map displaying and vehicle monitoring. The GVPS also performs high-volume transactions for clients accessing its resource components. The original GVPS architecture creates all map objects upfront whenever a new vehicle is added to the map rather than on-demand. This results in many unnecessary navigational threads consuming system resources. This in-turn impacts adversely on the GVPS performance. There is need for a better resource-aware configuration to manage object creation and method invocation in the GVPS. The current GVPS configuration also offers poor security features. It provides little access and authentication control for the transactions between client and resource components. Lastly, the current configuration offers little flexibility as it has strong coupling between its components.

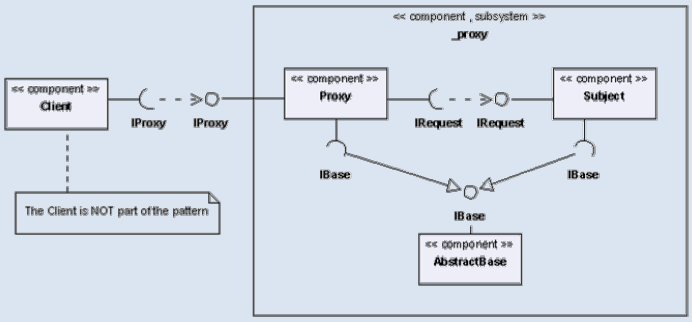
The mapping of the GVPS analysis scenario onto architectural design templates generated two possible architectural solutions: the *ClusterServer* pattern and the *Proxy* pattern. The design template detail descriptions are provided in Appendix F. Table 6.4 shows the contributions of the recommended design templates. Based on their contributions to the GVPS quality concerns, the *Proxy* pattern was selected to refine GVPS architecture. Although, *Proxy* pattern has a *"Not Applicable"* entered for the reliability concern, it has a high contribution for efficiency, security and performance, and a medium contribution for flexibility. The *ClusterServer* pattern scores poorly for security and flexibility, and only moderately well for performance and efficiency. A detailed description of the *Proxy* pattern properties is shown in Table 6.5.

Table 6.4. Architectural design alternatives contributions

Concerns	**Sub-Concerns**	**Architectural Design Alternatives**	
		ClusterServer	***Proxy***
Efficiency	Memory	Medium	High
	Processor	Medium	High
Flexibility	Expendability	N/A	Medium
Reliability	Availability	High	N/A
Security	Integrity	Low	High
Performance	ResponseTime	Medium	High

Table 6.5 *Proxy* pattern template

Category	Pattern
Name	*Proxy*
Also-Known-As	*Surrogate*
Related-Rules	*Decorator, Adapter*
Intent	The pattern makes the clients of a component communicate with a representative rather than to the component itself. Introducing such a placeholder can serve many purposes, including enhanced efficiency, easier access and protection from unauthorised access.
Context	Proxy is applicable whenever there is a need for more versatile or sophisticated reference a component. Some common situations in which the pattern is applicable: 1. Remote proxy – where clients of remote components should be shielded from network addresses and inter-process communication protocols. 2. Protection proxy – where components must be protected from unauthorised access 3. Cache proxy – where multiple simultaneous access to a component must be synchronised 4. Counting proxy – where accidental deletion of components must be prevented or usage statistic collected 5. Virtual proxy – where the processing or loading of a component might costly, while partial information about the component might be sufficient

	6. Firewall Proxy – where local clients should be protected from the outside world
Motivation	One reason for controlling access to a component is to defer the full cost of its usage until we actually need it. Until that point we can use some light objects (proxies) exposing an identical interface as the heavy objects to the Client. When the proxy is accessed it forwards the request to the real subject. This ability to control the access to a component can be required for a variety of reasons: caching, access control, synchronisation, lazy creation, remote access.
Configuration	
Consequences	Efficiency.Memory = {The proxy provides space optimisation through caching and lazy construction when the cost of data access and rendering is reduce, **H**} Efficiency.Proccesor = {The proxy provides time optimisation through caching and lazy construction when the cost of data access and rendering is reduce, **H**} Performance.ResponseTime = {A virtual proxy helps to implements a 'load-on-demand strategy' that avoid unnecessary loads and usually speeds up the application, however complex implementation would cause less efficiency due to indirection, **M**} Reusability.Modularity = {The proxy provides weak coupling between clients and subsystems, **M**} Flexibility.Expendability = {A remote proxy decoupling clients from the locations of remote server components, **H**} Security.Integrity = {Protection proxy and smart references allow additional housekeeping tasks when a component is accessed, **H**}

There is strong rationale for selecting the *Proxy* pattern [Buschmann96,Khosravi04]. The pattern can be implemented as a virtual or protection proxy to improve performance, security and enhance the functionality of the GVPS, it can also be implemented to create resource-hungry objects on demand to manage system resources. The *Subject* component i.e. *Map* and *avpsDB* is a resource-hungry component that we wish to use more efficiently. The *proxy* component acts as a surrogate, holding a private instance of a *subject* component as required. The client components, *IVD_Console* and *CC_Console,* execute actions on the *proxy* whose results are passed to the *Subject* component. The results from the *Subject's* members are returned to the client via the *proxy*. The *AbstractBase* component is shared by the *proxy* component and its *subject* component. The *base*

component defines any standard members that will be implemented by *proxy* component and *subject* components. Therefore, the virtual proxy can effectively delay the creation of a rich environment. Secondly, the *proxy* enhances access security by ensuring that only authenticated components can access the database. The protection proxy component acts as a layer between these components and the database.

6.3.4 Refining GVPS Architecture

Based on the *Proxy* pattern, the original GVGS architecture (see Fig. 6.3) was revised as shown in Fig. 6.6. The modifications, which also involve mapping the GVPS services to on the *Proxy* pattern components, are shown in the boxed area.

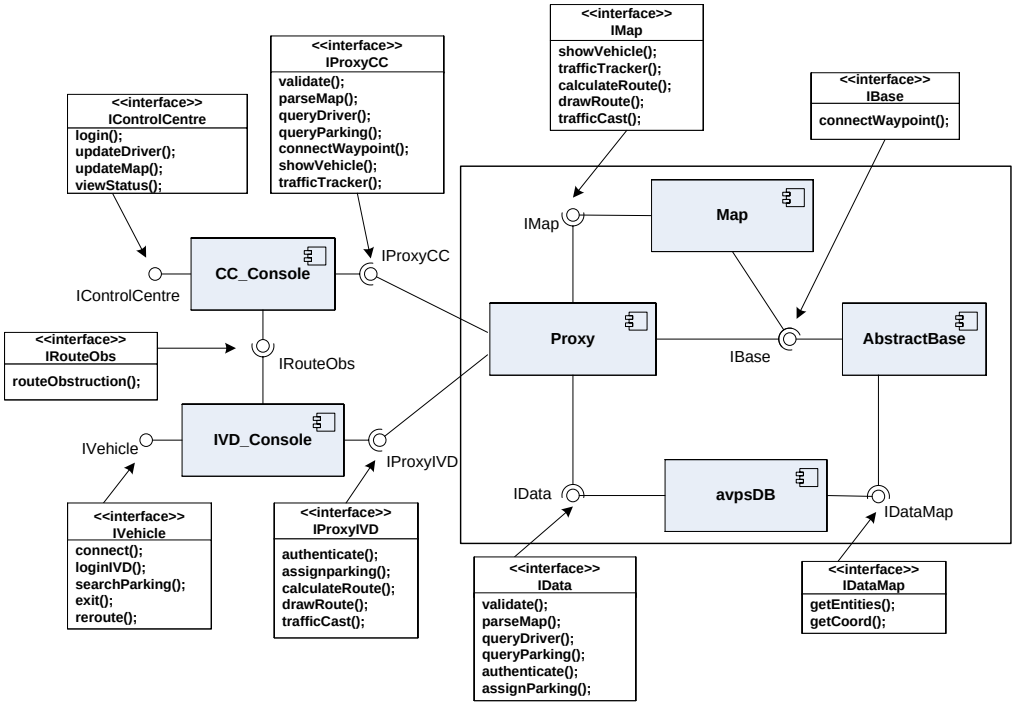


Fig. 6.6 *Proxy* pattern (S2)

The visualisation of the mapping process is shown in Fig. 6.7. The services *ValidationAdmin*, *ParseMap*, *ManageParking* and etc. are mapped onto *IProxy* and

IProxyIVD of *Proxy* abstract component. The mapping to concrete components was conducted as described in Chapter 4 (Section 4.1.4).

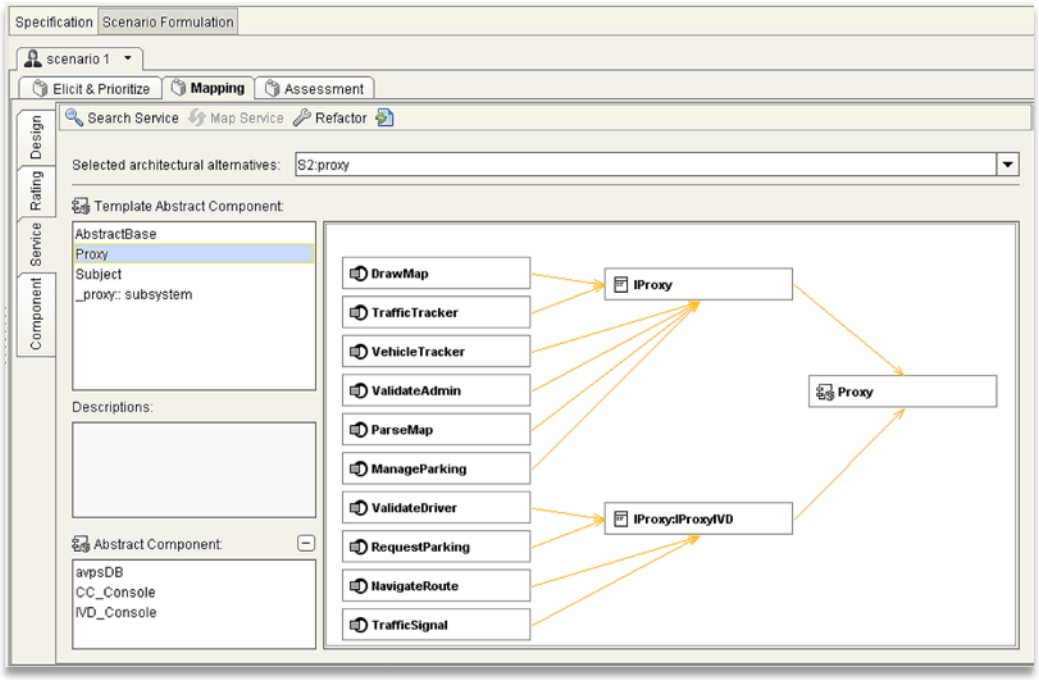


Fig. 6.7 *Proxy* mapped services onto *IProxy* and *IProxyIVD*

The weighted contributions of the two design alternatives[1] are shown in Fig. 6.8. The *Proxy* pattern alternative (S2) offers an overall quality contribution score of 0.824 (i.e. efficiency 0.26, flexibility is 0.13, performance is 0.17, reliability is 0.00 and security 0.26). The *ClusterServer* pattern (S1) offers lower contribution score of 0.521 (i.e. efficiency 0.17, flexibility is 0.00, performance is 0.17, reliability is 0.09 and security 0.9). The S1 and S2 contributions are further refined to show their contributions at sub-concern level (see Fig. 6.9).

[1] Details weighting and scoring values are compiled in Appendix E3, Table E3.1

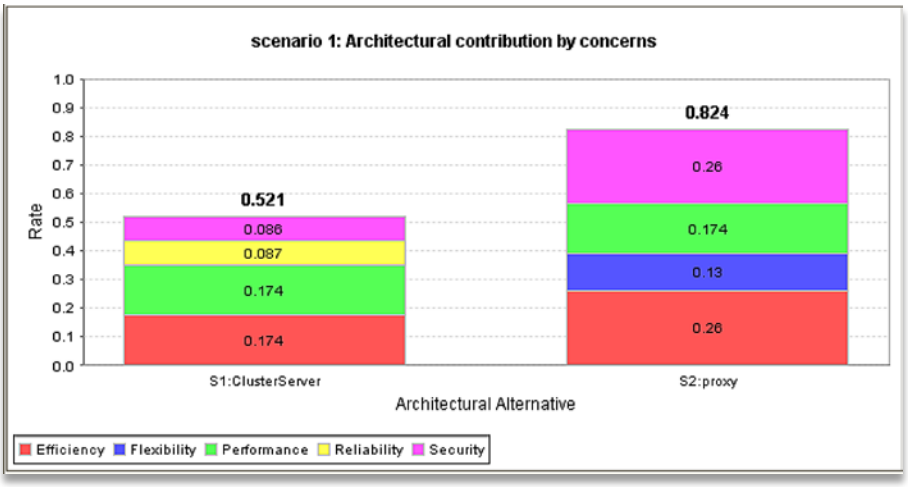


Fig. 6.8 Contribution of suggested alternatives according to main concerns

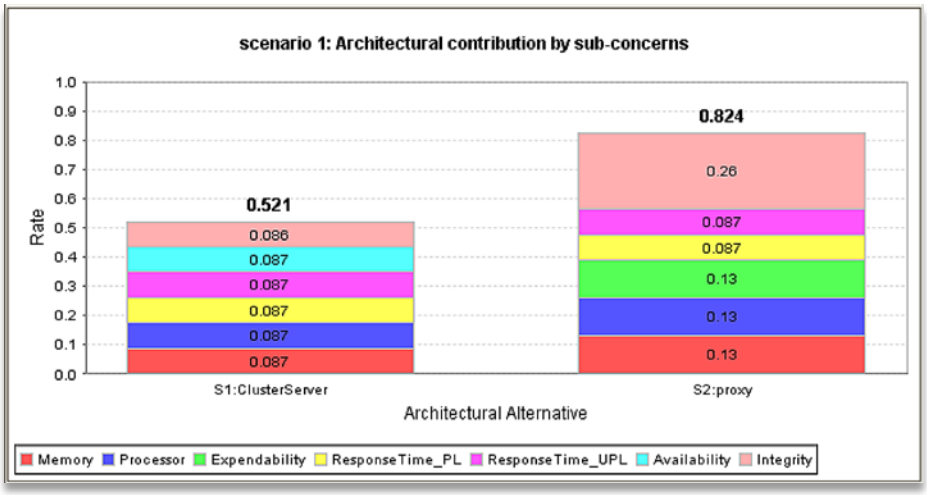


Fig. 6.9 Contribution of suggested alternatives according to sub-concerns

A sensitivity analysis completes the analysis by examining the robustness of the selected architectural solution to changes in the identified quality concerns. A software designer may, for example, be concerned about the weight of security (i.e. 0.26) relative to efficiency (i.e. 0.26), flexibility (i.e. 0.13), performance (i.e. 0.26) and reliability (i.e. 0.09), and might want to know how changes in these weights might affect the contributions of the alternative designs. Fig. 6.10 shows how the value of benefits for the design alternatives varies with changes in security. If security had a weight of zero, this would imply that the two security sub-concerns would also have zero weights.

After re-normalisation, weights would be 0.35 for efficiency, 0.35 for performance, 0.18 for flexibility and 0.12 for reliability. This would cause S2 to have an aggregate benefit value of 0.765. If security had a weight of 100 (and therefore efficiency, flexibility, performance and reliability a weight of zero) S2 would have an aggregate benefit value of 1.0. Sensitivity analysis conducted for security concern as shown in Fig. 6.10 indicate that the *Proxy* pattern is the more viable alternative design. The *ClusterServer* pattern may be considered when security concern is zero, or less important, as its contribution only slightly lower than that of the *Proxy* pattern. However, the uncertainty of using *Proxy* pattern reduces when the weight of the security concern increases.

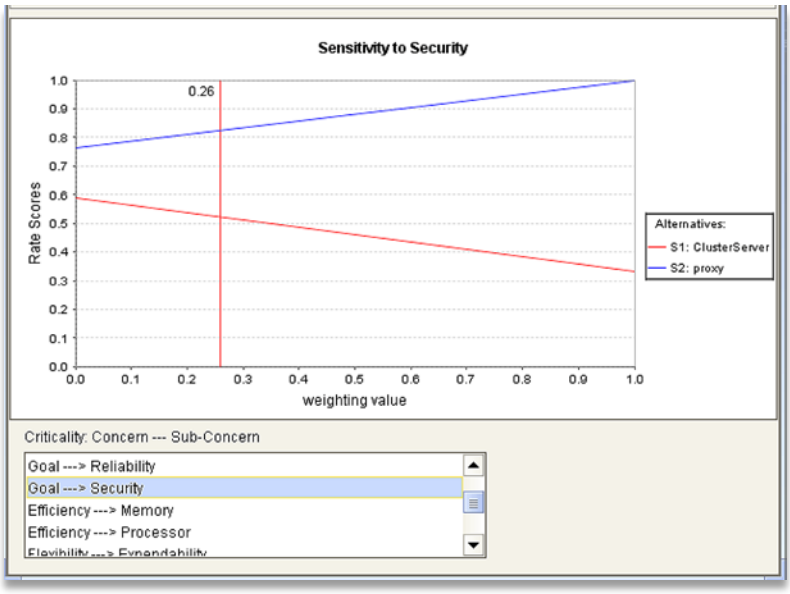


Fig. 6.10 Sensitivity analysis applied to security concern

6.4 Runtime Comparison of GVPS Architectures

This section describes an experiment to compare the runtime performance and resource consumption of the original and refined GVPS architectures. The architectures were implemented to simulate the GVPS in operation. Each implementation comprised the In-Vehicle-Device (IVD) sub-system and the Control

Centre. The GVPS implementation of the original architecture was named Simulator I and the refined architecture implementation, Simulator II. The simulators were implemented using JavaBeans technology [Java10] and constructed according to the architectures shown in Fig. 6.3 and Fig. 6.6.

The basic GUI design for the GVPS simulators is shown in Fig. 6.11–Fig. 6.13. However, Simulator II implements more functionality as it addresses more GVPS requirements [Cs10] than Simulator I. For example, Simulator II allows parking spaces to be allocated by vehicle type (e.g. car, disabled and van/lorry) and user type (e.g. Staff, Student, Visitor), whereas Simulator I allocates spaces only according to vehicle type. Simulator II also improves the map display by labelling both buildings and parking areas as shown in Fig. 6.13.

Fig. 6.11 The GVPS simulator main window

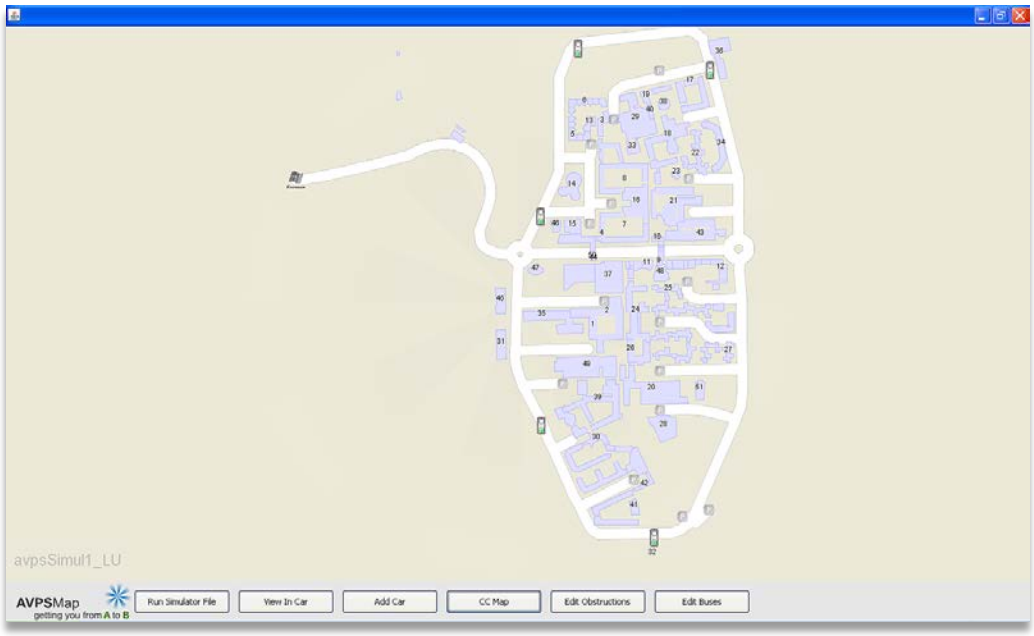


Fig. 6.12 Simulator I display Lancaster University map with 'avpsSimul1_LU' tag on left bottom panel

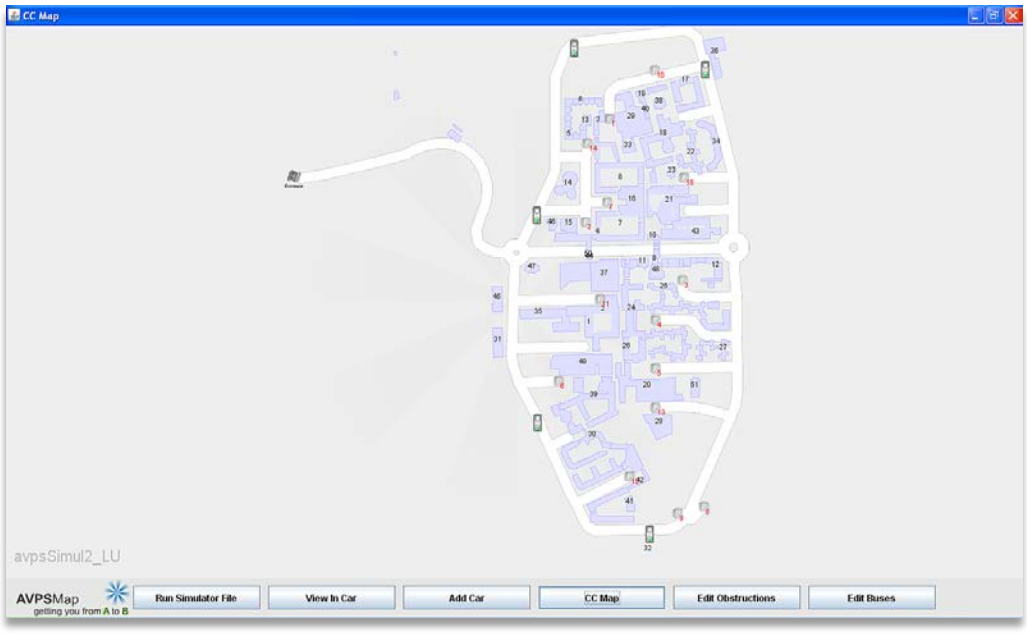


Fig. 6.13 Simulator II display Lancaster University map with 'avpsSimul2_LU' tag on left bottom panel

6.4.1 Methodology

The experiments were conducted using Java VisualVM [Java11]. The two simulators were configured to run the same data comprising different numbers of vehicles and different road conditions. The simulations were tracked using the basic VisualVM runtime information such as process id (PID), their main class, arguments passes to java process, JVM version, JDK home, JVM flags and arguments, and system properties.

Six experiments, representing three data scenarios, were conducted for each simulator. Each experiment observed the system behaviour when vehicles entered the campus in search of parking spaces (entering event), and when vehicles left their parking spaces to exit the campus (exiting event). Data on performance and resource consumption (i.e. memory usage, CPU time, heap memory, number of loaded classes) and the number of threads running during entering and exiting events was collected and analysed for each scenario. The memory profiler and CPU profiler were used to assess where the application spend most time and which objects consumed most memory during the entering and exiting events.

Experiment Scenario 1: One student vehicle and one visitor vehicle under normal road conditions

The objective of this experiment was to compare the behaviour of the two architectures under relatively low load conditions with normal road conditions (i.e. no road closures). The experiment scenario consisted of running an auto-navigation file that specified a student vehicle of type *car* and a destination of the InfoLab 21 (see Fig. 6.14 – Fig. 6.16), then adding a visitor vehicle of type car whose destination was the Sport Centre (see Fig. 6.17). After a short while, the vehicles exited from their parking areas and proceeded to leave the campus.

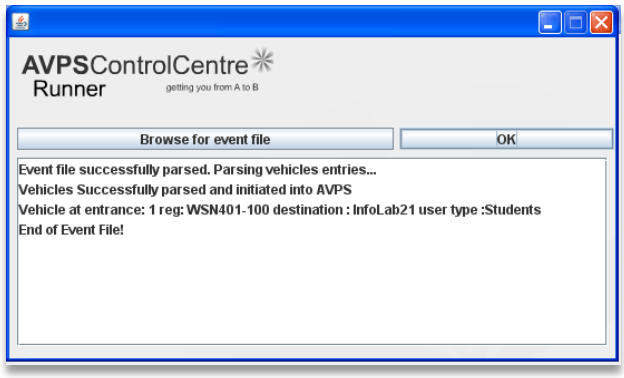


Fig. 6.14 Navigation event for a student car to Info Lab 21

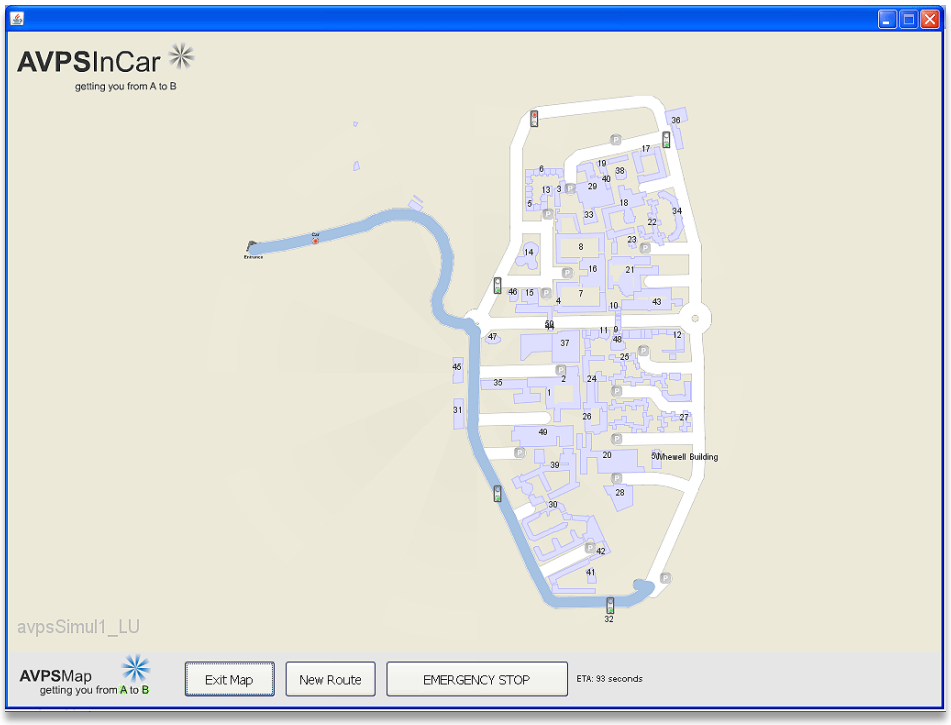


Fig. 6.15 Student car navigates to Info Lab 21 parking area shows on IVD panel of Simulator I

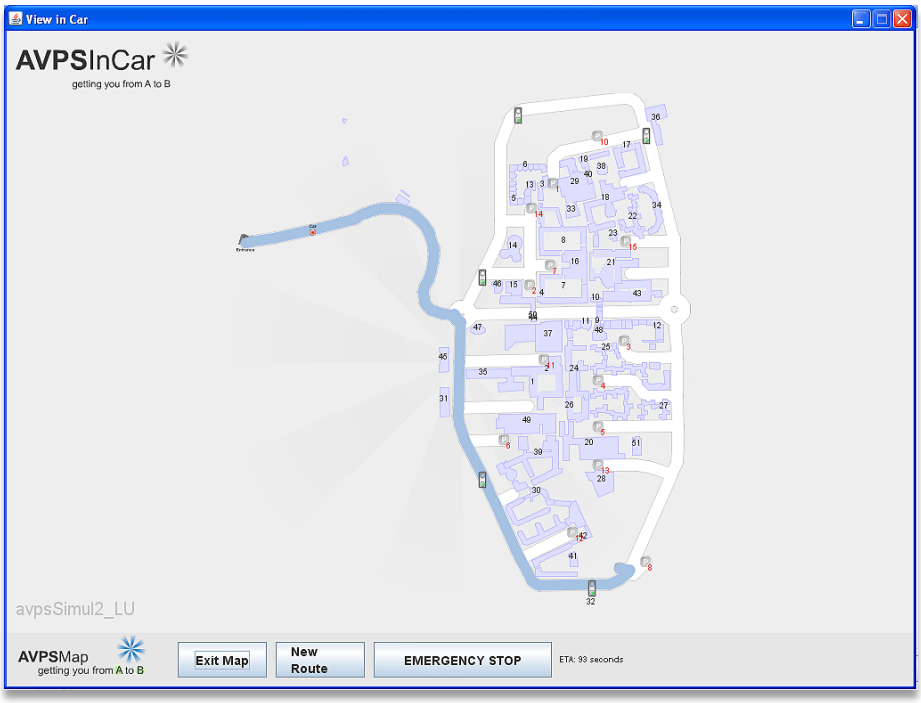


Fig. 6.16 Student car navigates to Info Lab 21 parking area shows on IVD panel of Simulator II

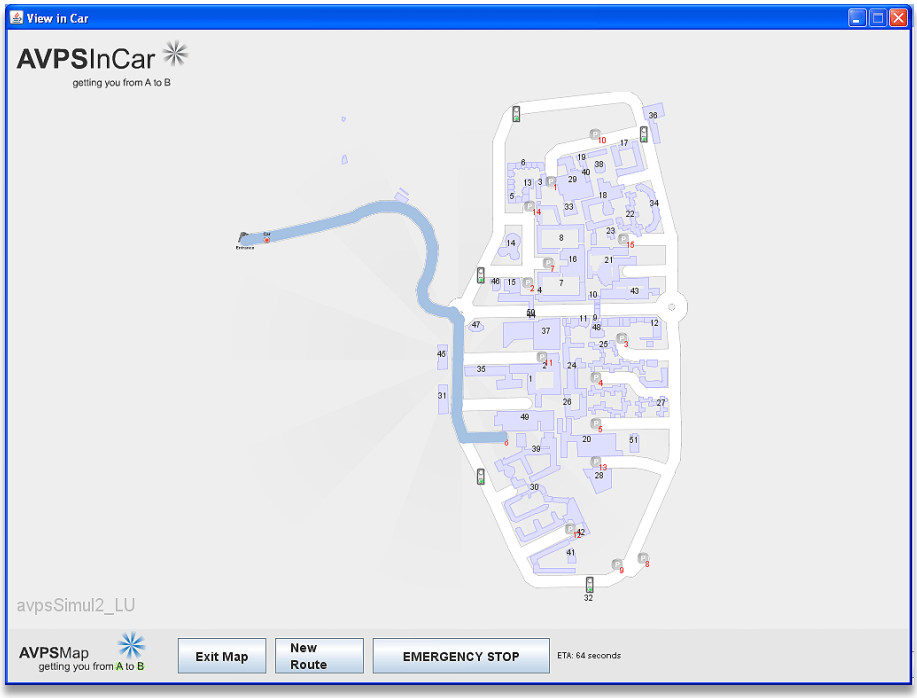


Fig. 6.17 Visitor car navigates to Sport Centre parking area shows on IVD panel of Simulator II

Fig. 6.18 and Fig. 6.19 show Simulator I and Simulator II executing Experiment Scenario 1. Simulator I is packaged as avpsSimulator1.jar and its main method is located in the *avpscc* bean. Simulator II is packaged as avps3.jar and its main method is located in the *ControlCentre* bean. Both simulators are running on a local host and using the Java Runtime Environment (JRE) version 6. Detailed system properties for the simulators are displayed on the right lower panels.

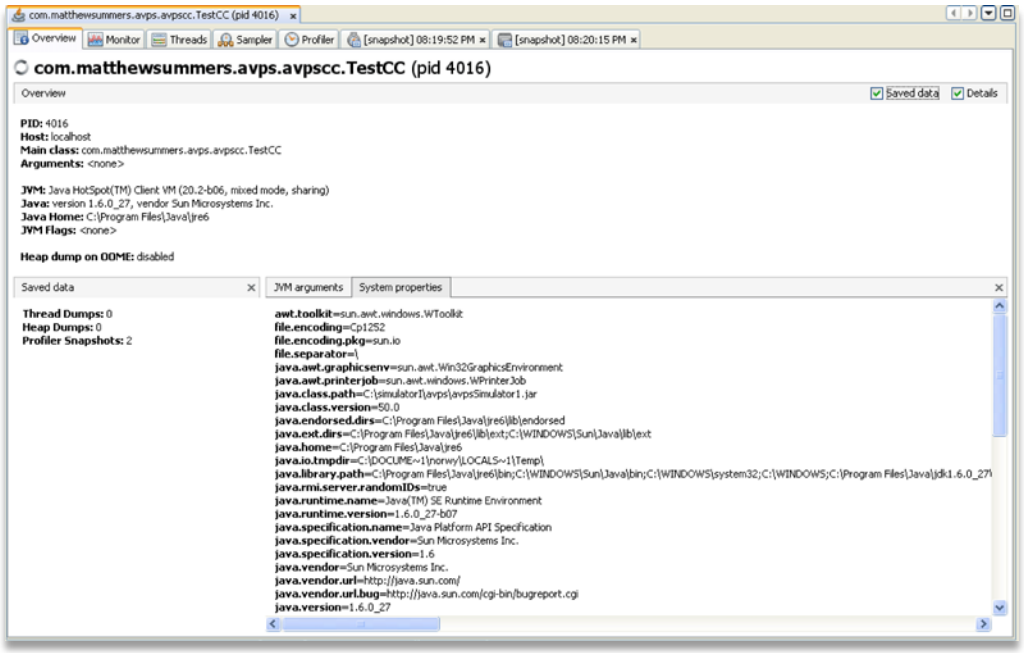


Fig. 6.18. Simulator I (PID 4016) configurations and environment

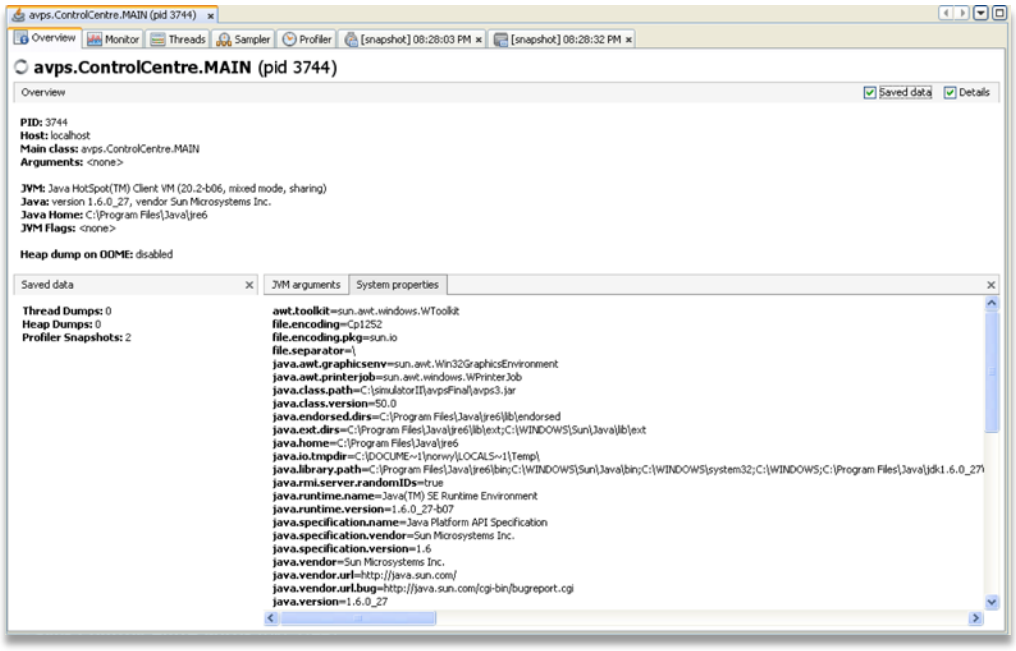


Fig. 6.19 Simulator II (PID 3744) configurations and environment

Performance and resource analysis for Experiment Scenario 1

The result of monitoring the performance and resource consumption of the two simulators is shown in Fig. 6.20 and Fig. 6.21. The left upper panel shows the percentage of CPU time used (orange line) and the garbage collector (GC) activity (blue line). The heap graph located in the right upper panel shows information on memory consumption and memory pools. The memory used includes the memory occupied by all objects including reachable and unreachable objects. The used area turns red when the memory used exceeds the memory usage threshold. The Heap graphs shows memory usage for current heap size, which indicates number of Kbytes currently, occupied by the heap and maximum heap size that indicates the maximum number of Kbytes occupied by the heap.

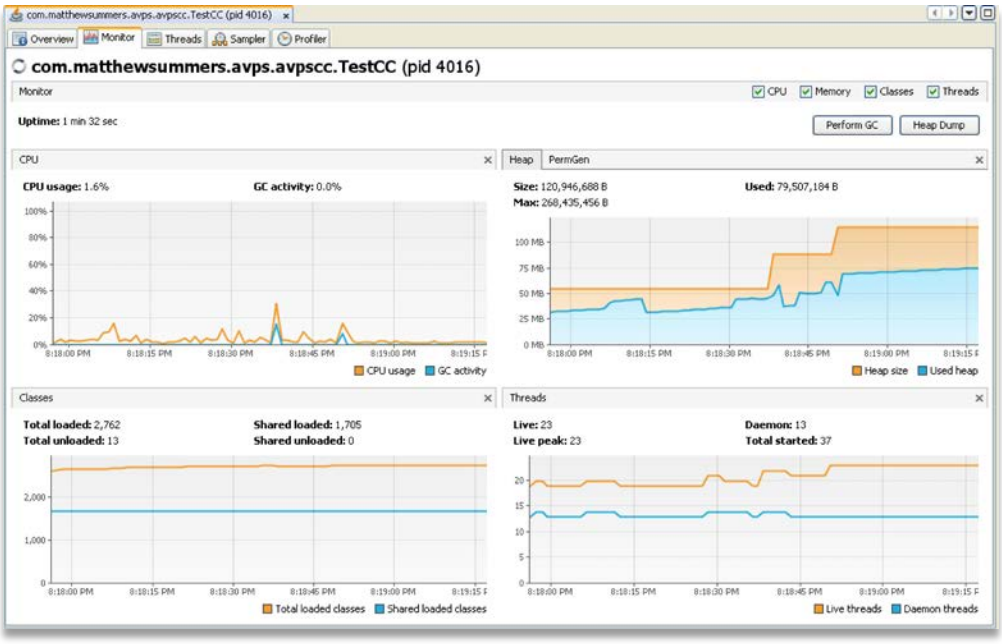


Fig. 6.20 Simulator I (PID 4016) monitor entering event

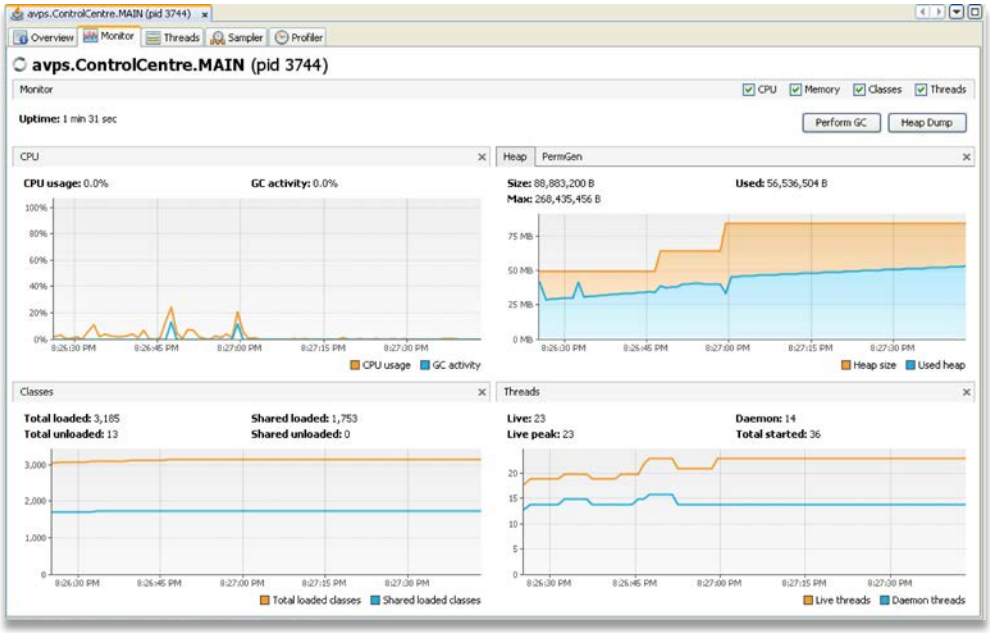


Fig. 6.21 Simulator II (PID 3744) monitor entering event

The classes graph located in the right lower panel displays an overview of the total number of classes loaded in memory (orange line) and the shared classes (blue line) versus time. The total classes loaded indicate the total number of classes loaded into

memory since JVM started, including those subsequently unloaded. Lastly, the total classes unloaded represents the number of classes unloaded from memory since the JVM started. The Threads graphs located on the left lower panel provides an overview of the number of live and daemon threads versus time in the application's JVM. The threads graph (orange line) represents the current number of live daemon threads plus non-daemon threads. The blue graph indicates the current number of live daemon threads and total number of threads started since JVM started (i.e. daemon, non-daemon, and terminated).

The CPU, heap, classes and thread graphs for Simulator I (see Fig. 6.20) and Simulator II (see Fig. 6.21) correspond to the entering event. Simulator I shows that this event uses 1.6% of CPU time and 79,507,184 bytes of memory while loading 2,762 classes and has a total of 23 live threads. Conversely, Simulator II uses 0.0% of CPU time and 56,536,504 bytes while loading more classes; 3,185 classes and a total of 23 live threads. In the heap graphs, the memory consumption shows two expected spikes. The first spike occurs after running the auto-navigation file for the student vehicle and the second spike occurs after adding a new vehicle.

At approximately 360 seconds, the exiting event is triggered. The results show that Simulator I uses 3.0% of CPU time and 67,933,744 bytes of memory, which involves 4,622 classes and 27 live threads (see Fig. 6.22). Simulator II shows a markedly better performance and significantly less memory consumption for the same event. It is also worth mentioning that Simulator II is running 13% more classes than Simulator I. Simulator II uses 1.6% of CPU time and 44,644,992 bytes of memory while loading 5,237 classes and 26 live threads (see Fig. 6.22). In the heap graphs memory consumption shows two expected spikes. The first spike occurs when the first vehicle exits its parking area and the second when the second vehicle exits its parking area.

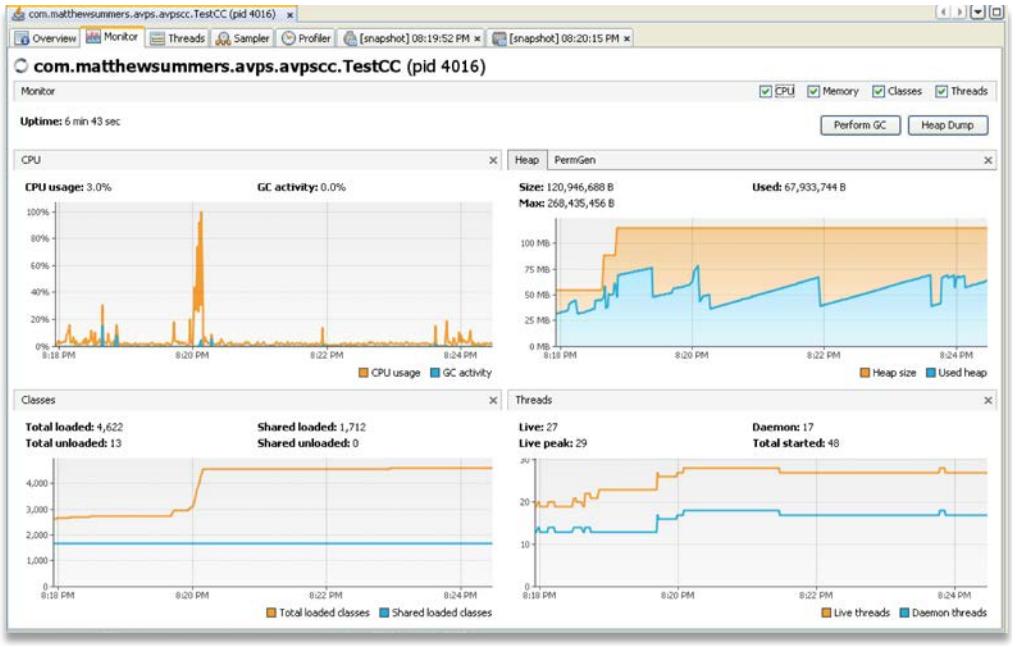


Fig. 6.22 Simulator I (PID 4016) monitor exiting event

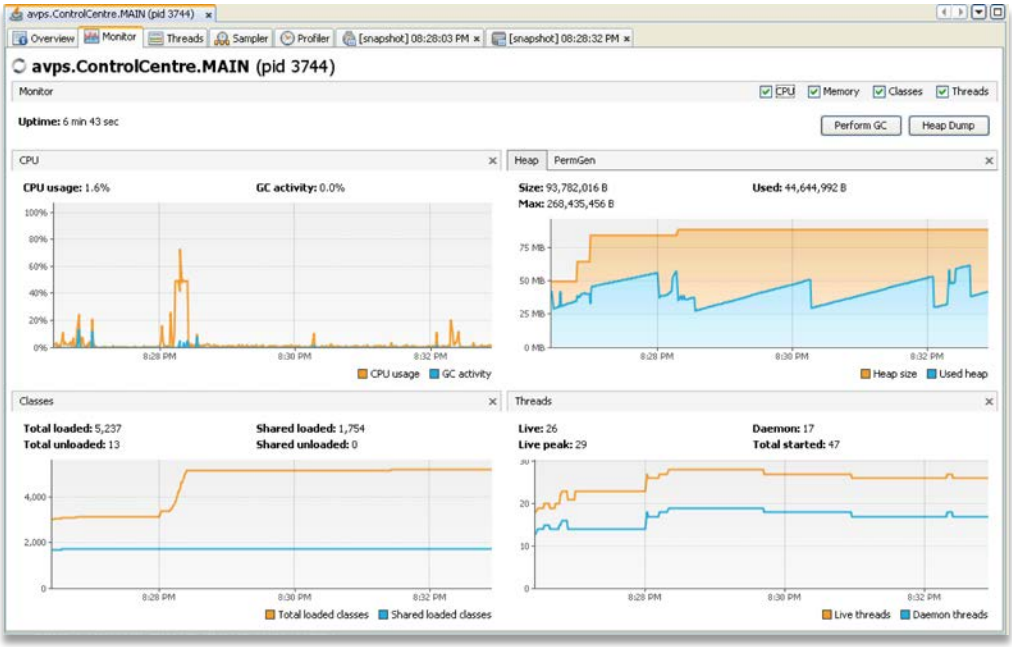


Fig. 6.23 Simulator II (PID 3744) monitor existing event

The performance and memory consumption results for Simulator I and Simulator II in Experiment Scenario 1 are summarised in Table 6.6. The results show that

Simulator II performed significantly better than Simulator I. Although Simulator II ran more classes and threads, Simulator II has significantly lower CPU usage and memory consumption than Simulator I. The proxy design pattern adopted for the refined GVPS architecture manages complex and resource hungry components such as *Map* and *avpsDB* more effectively than the original GVPS architecture.

Table 6.6 Summary performance and memory consumption for Experiment Scenario 1

		CPU Usage	Heap (bytes)	Classes	Threads
Entering event	Simulator I	1.6%	79, 507, 184	2, 762	23
	Simulator II	<0.0%	56, 536, 504	3, 185	23
Exiting event	Simulator I	3.0%	67, 933, 744	4, 622	27
	Simulator II	1.6%	44, 644, 992	5, 237	26

CPU and memory profiles for Experiment Scenario 1

Profiling the simulators is valuable for exploring where the application spends most of its time and for establishing which objects consume most memory. Profiling can also expose potential memory leaks. Fig. 6.24 and Fig. 6.25 show the CPU profile for Simulator I and Simulator II in Experiment Scenario 1 during entering event. The profile lists all the methods called during the event and the time consumed by the methods. Simulator I shows that AWT-EventQueue-) the application spends almost all of its time on three methods. Simulator II (Fig 6.24) shows that for Thread-5 and AWT-EventQueue-0, which have several methods, only the one method run 100% of the time.

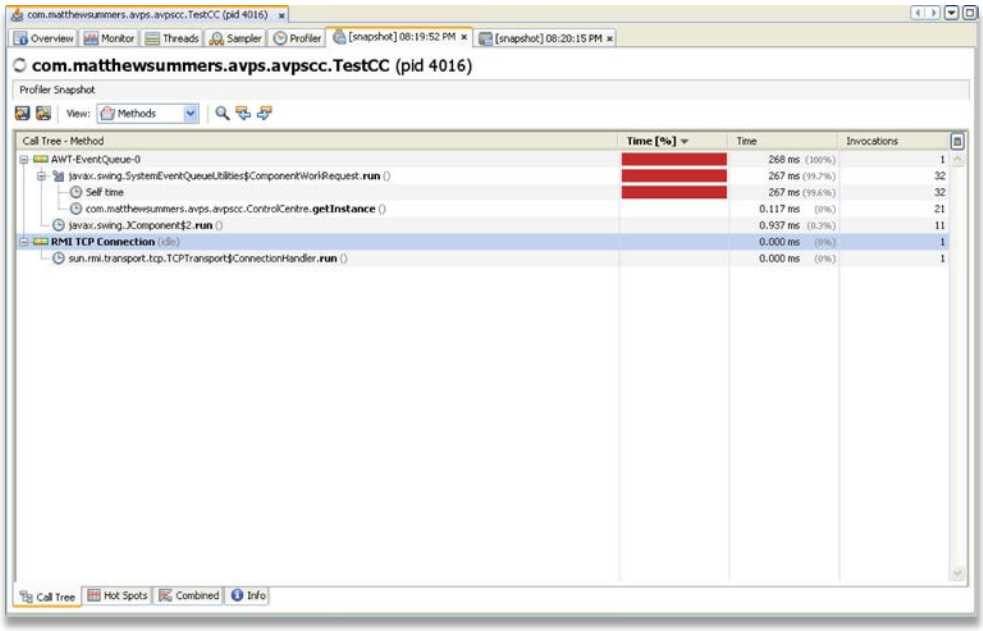


Fig. 6.24 Simulator I (PID 4016) CPU profile

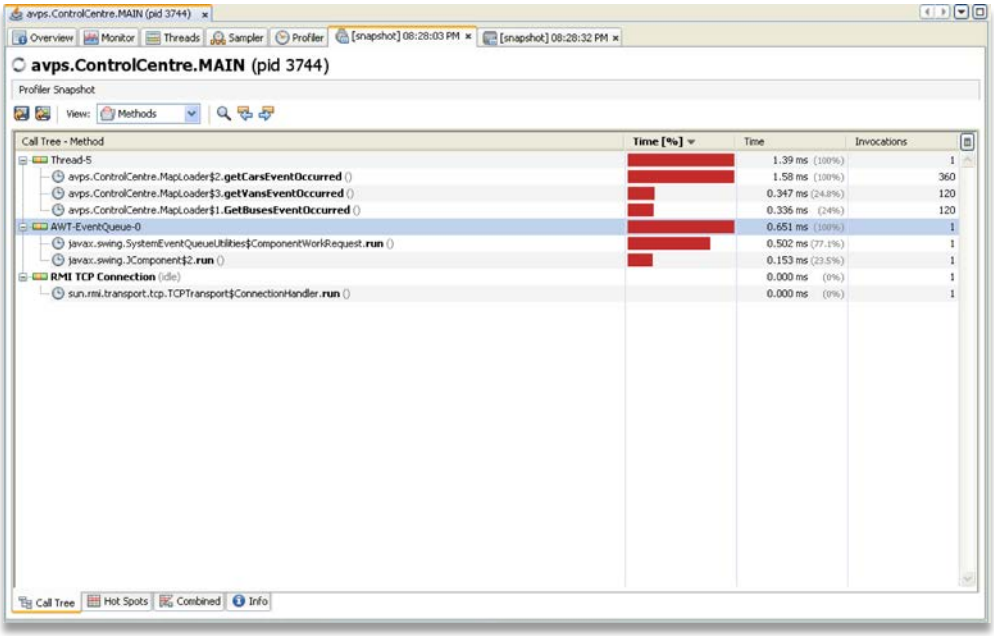


Fig. 6.25 Simulator II (PID 3744) CPU profile

Memory profiles for the entering event, for Simulator I and II in Experiment Scenario 1, are shown in Fig. 6.26 and Fig. 6.27. The profiles provide detailed memory consumption for all objects involved in the entering event. In Simulator I, the

highest memory consumption is 17.9%, for object char[]. In Simulator II the highest memory consumption for object char[] is 16.5%. All the other objects in Simulator II consume significantly less memory than the objects in Simulator I.

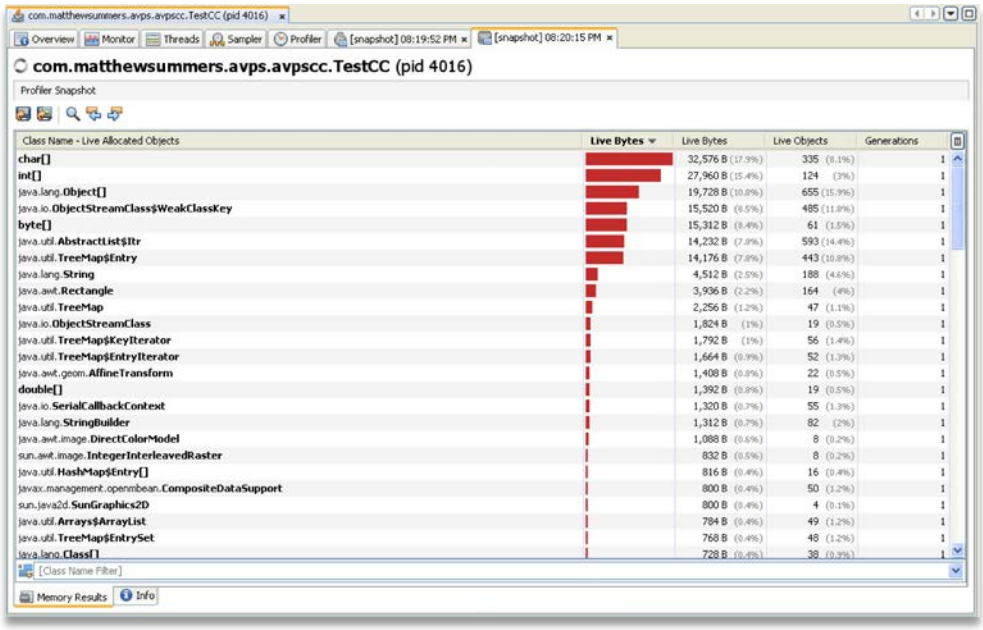


Fig. 6.26 Simulator I (PID 4016) Memory profile

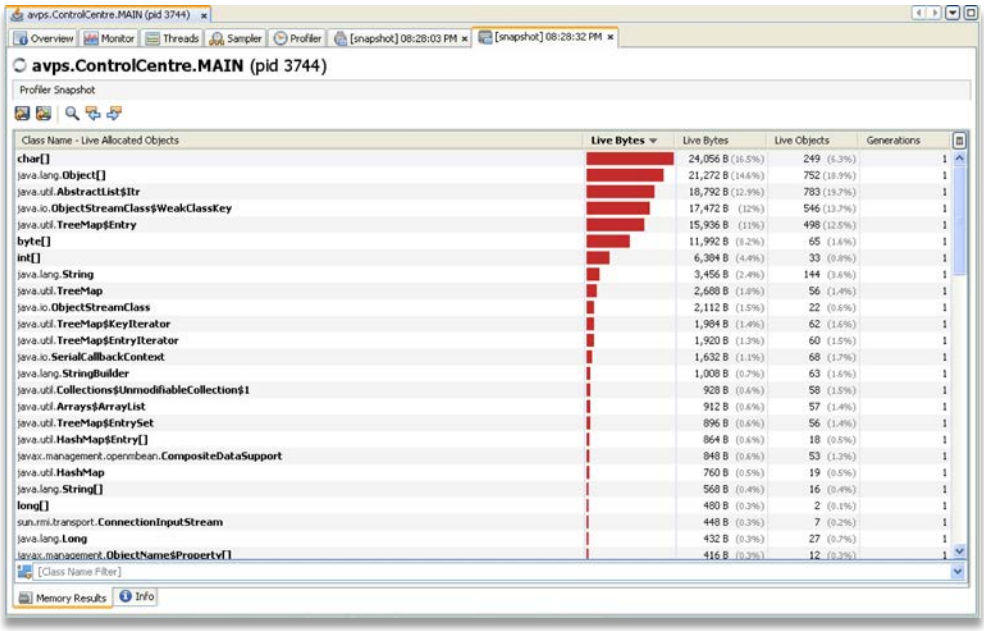


Fig. 6.27 Simulator II (PID 3744) memory profile

Experiment Scenario 2: Two student vehicles and three visitor vehicles under normal road conditions

The objective of the second experiment was to compare the behaviour of the two architectures under increased load conditions (i.e. 2.5 times the load in Experiment Scenario 1). The experiment used five vehicles; two student vehicles and three visitor vehicles under normal roads conditions. The student vehicles of type car navigated to the Ash House (see Fig. 6.28 and Fig. 6.29). Two visitor vehicles navigated to the Ruskin Library (see right side of Fig. 6.30) and one visitor vehicle navigated to the Health Centre (see left side of Fig. 6.30). After a short while, the vehicles exited from their parking areas and proceeded to leave the campus (see Fig. 6.31).

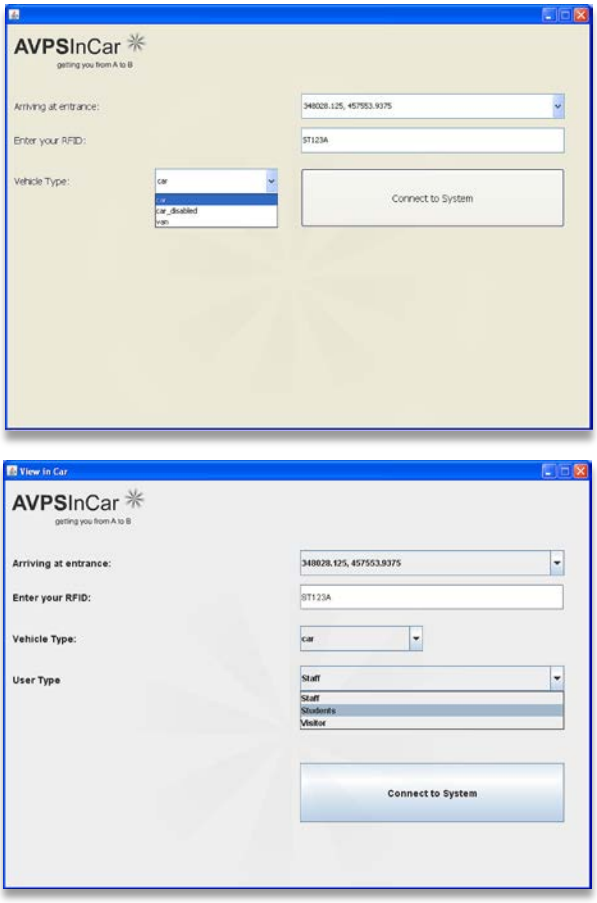


Fig. 6.28 Student car arrives at the university entrance on IVD panel of Simulator I (top) and Simulator II (bottom)

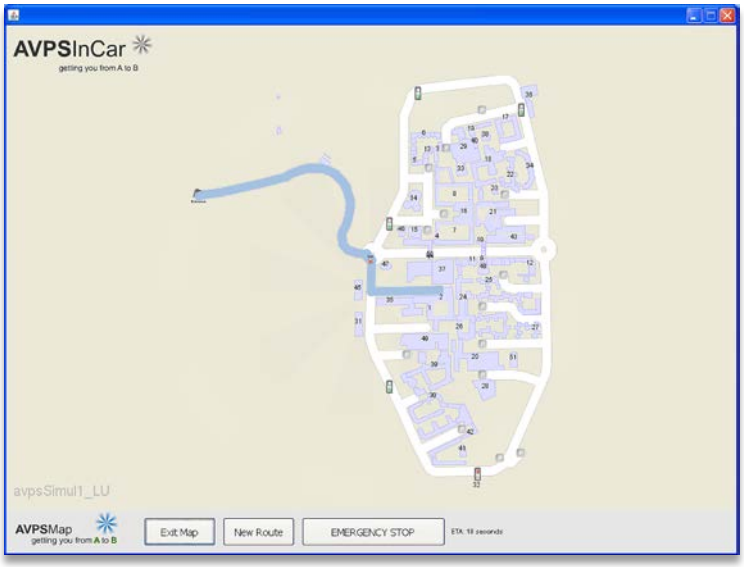


Fig. 6.29 Student car navigates to Ash House parking area shown on IVD panel of Simulator I

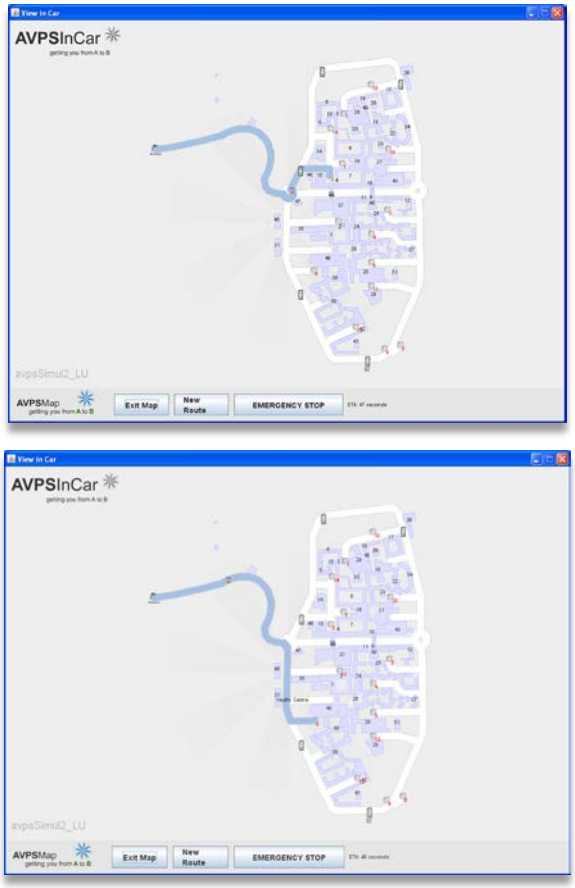


Fig. 6.30 The first visitor car navigates to Ruskin Library parking area and the second visitor car navigates to Health Centre parking area shown on IVD panel of Simulator II

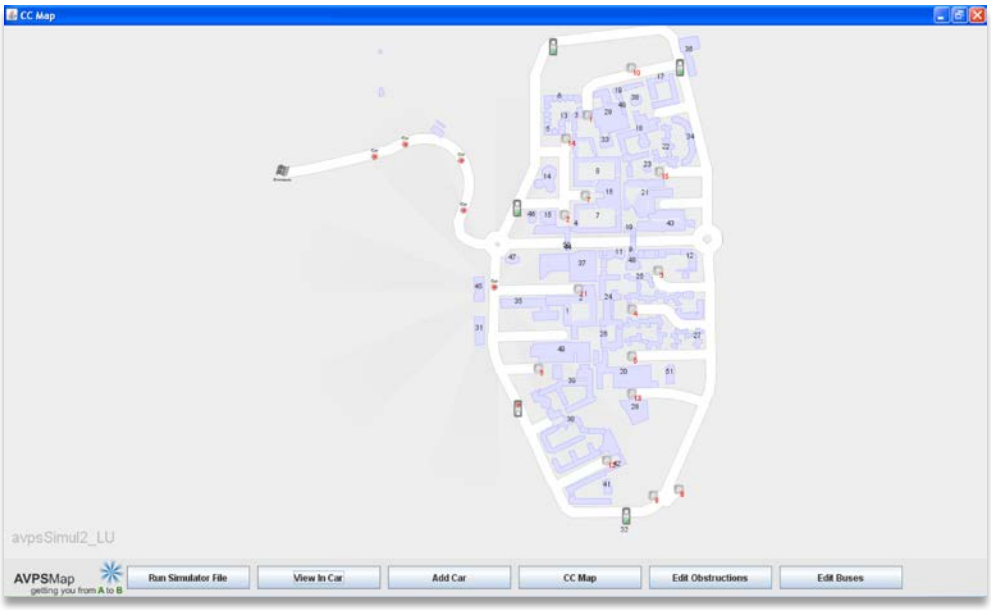


Fig. 6.31 Student and visitor cars leaving car park areas shown in Control Centre panel of Simulator II

Performance and resource analysis for Experiment Scenario 2

The VisualVM CPU, heap, classes and thread graphs, corresponding to the GVPS entering event, are shown in Fig. 6.32 (Simulator I) and Fig. 6.33 (Simulator II). Simulator I shows that the entering event uses 2.4% of CPU time and consumed 97,109,032 bytes of memory while loading 2,721 classes and has a total of 28 live threads. Conversely, Simulator II uses 0.7% of CPU time and consumed 92,927,384 bytes while loading more classes and threads; 3,175 classes and 30 live threads. In the heap graphs, the memory consumption shows five spikes that occurred after adding two student vehicles and three visitor vehicles. The result for both simulators in Experiment Scenario 2 shows more memory consumptions compared to Experiment Scenario 1, although the classes loaded are fewer. This may be due to the fact that both the simulators in Experiment Scenario 2 running more threads, which require more resources. Nevertheless, Simulator II still shows significantly better performance and memory consumption in Experiment Scenario 2 than Simulator I.

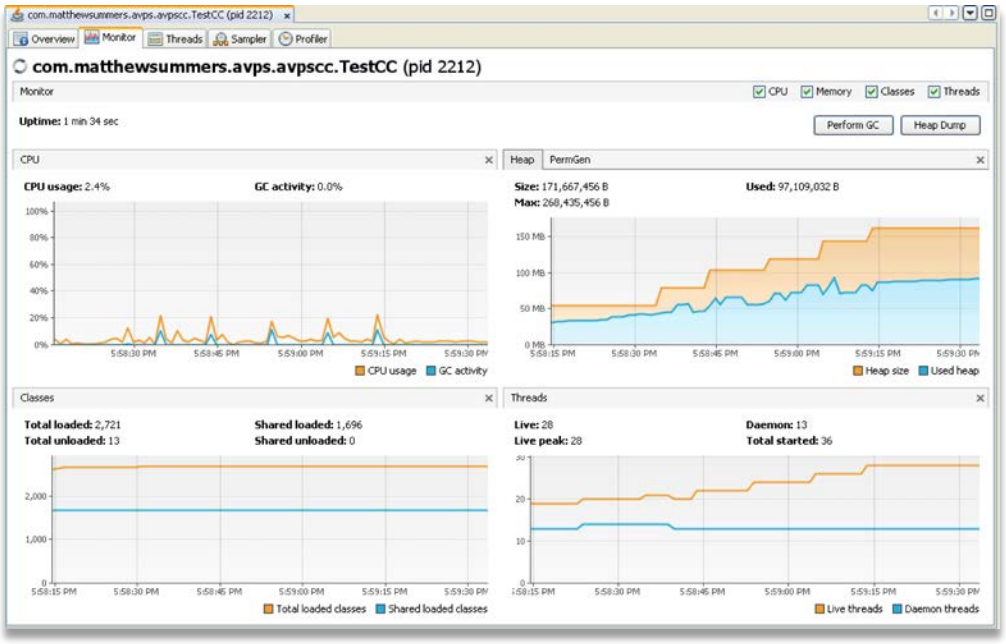


Fig. 6.32 Simulator I (PID 2212) monitoring entering event

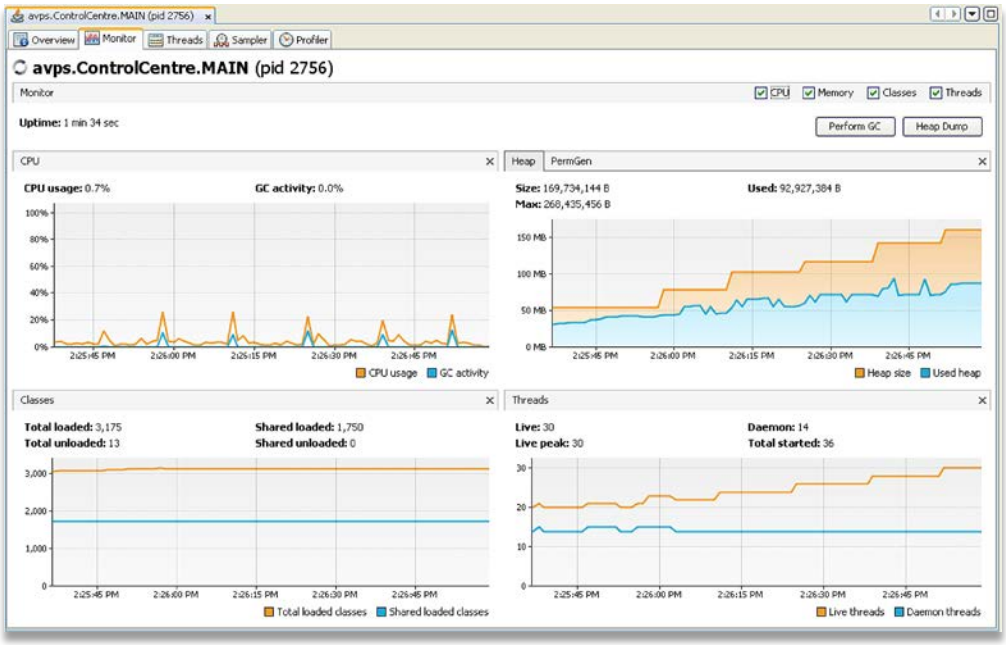


Fig. 6.33 Simulator II (PID 2756) monitoring entering event

In Experiment Scenario 2 the exiting event is triggered at approximately 750 seconds (see Fig. 6.34 and Fig. 6.35). The results show that Simulator I uses 2.2% of

CPU time and 100,625,744 bytes of memory that involves 4,556 classes and 33 live threads. Simulator II uses 1.5% of CPU time and 93,155,208 bytes of memory while loading 5,214 classes and 34 live threads. In the heap graphs, the memory consumption shows five spikes. These occur as the five vehicles exit their parking areas.

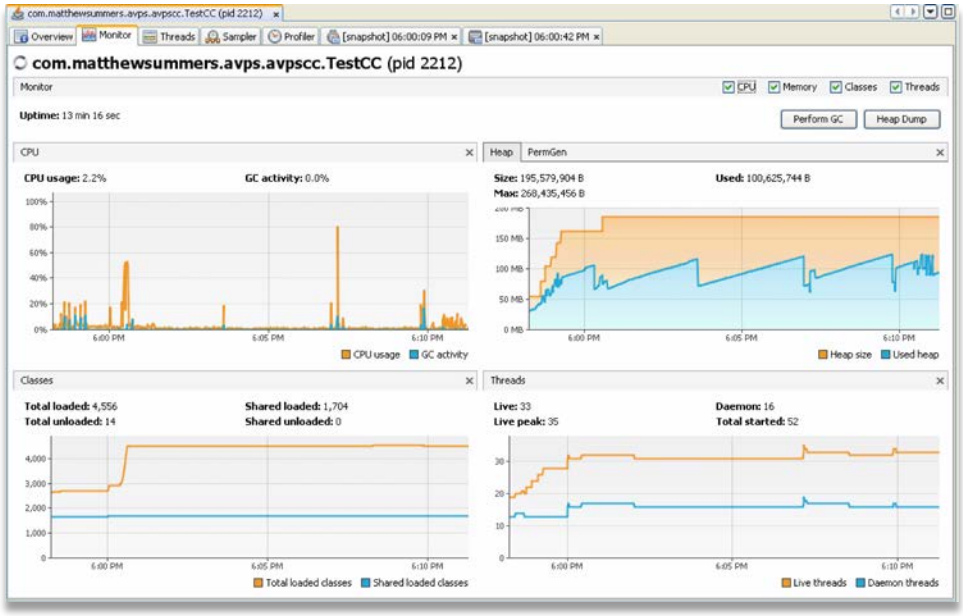


Fig. 6.34 Simulator I (PID 2212) monitoring exiting event

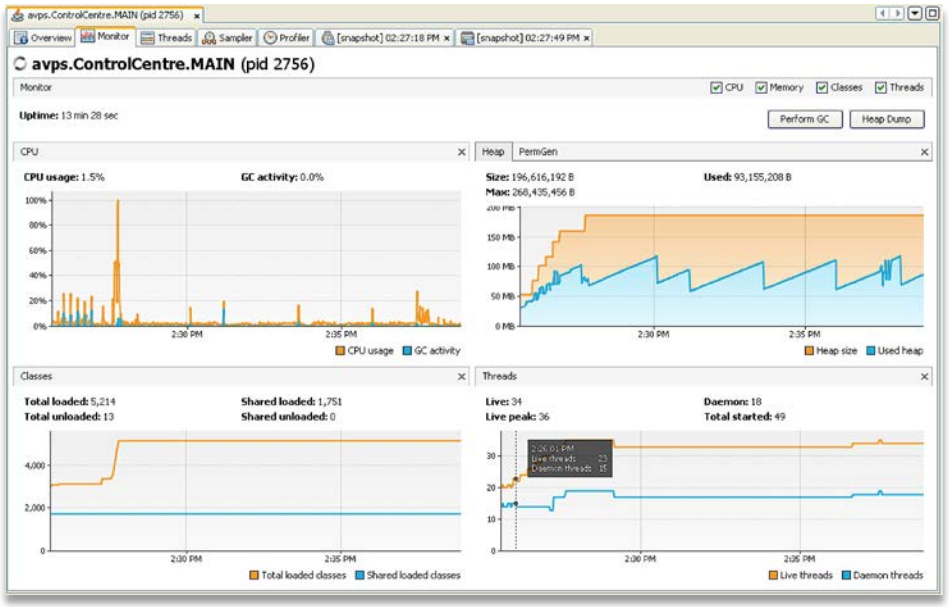


Fig. 6.35 Simulator II (PID 2756) monitoring exiting event

The performance and memory consumption results for Simulator I and Simulator II in Experiment Scenario 2 are summarised in Table 6.7. The results show that Simulator II performed significantly better than Simulator I. Although Simulator II run more classes and threads, it has lower CPU usage and memory consumption compared to Simulator I. In the entering event Simulator II uses almost 3.5 times less CPU time than Simulator I, and almost 5Mbytes less in memory. In the exiting event Simulator II uses 1.5 times less CPU time than Simulator I, and almost 7Mbytes less in memory

Table 6.7 Summary performance and memory consumption for Experiment Scenario 2

		CPU Usage	Heap (bytes)	Classes	Threads
Entering event	Simulator I	2.4%	97, 109, 032	2, 721	28
	Simulator II	0.7%	92, 927, 384	3, 175	30
Exiting event	Simulator I	2.2%	100, 625, 744	4, 556	33
	Simulator II	1.5%	93, 155, 208	5, 214	34

CPU and memory profiles for Experiment Scenario 2

The CPU and memory profiles during the entering event, for Simulator I and Simulator II, are shown in Fig. 6.36 and 6.37. Once again Simulator I shows that application spends almost all its time on a few methods in the AWT-EventQueue-are. In Simulator II, the application its time on several methods in the Thread-5, AWT-EventQueue-0 and RMI-TCP-Connection.

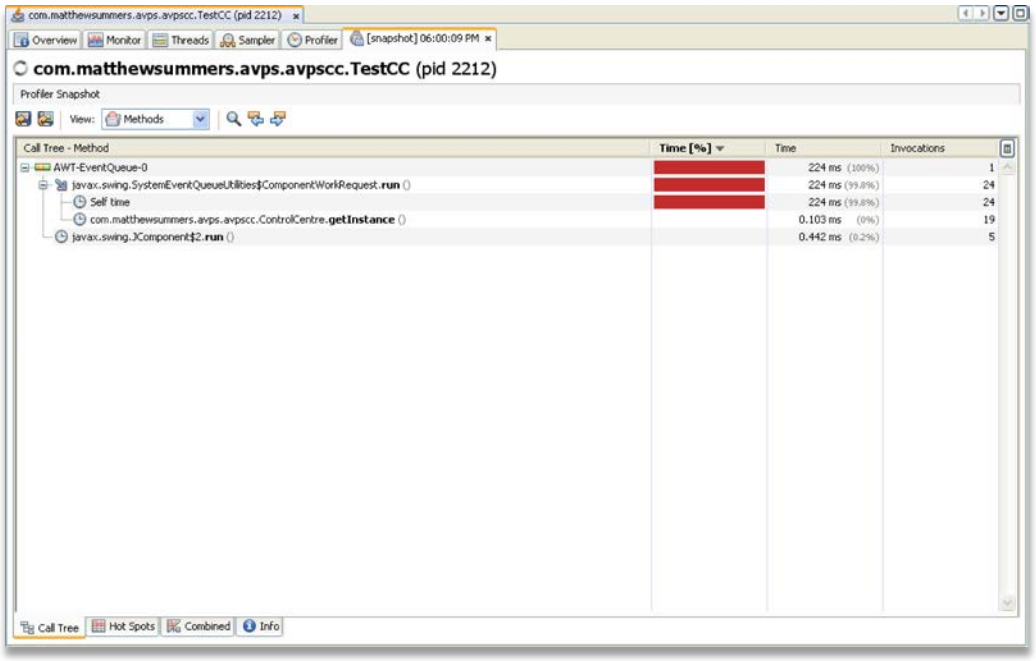


Fig. 6.36 Simulator I (PID 2212) profiling CPU

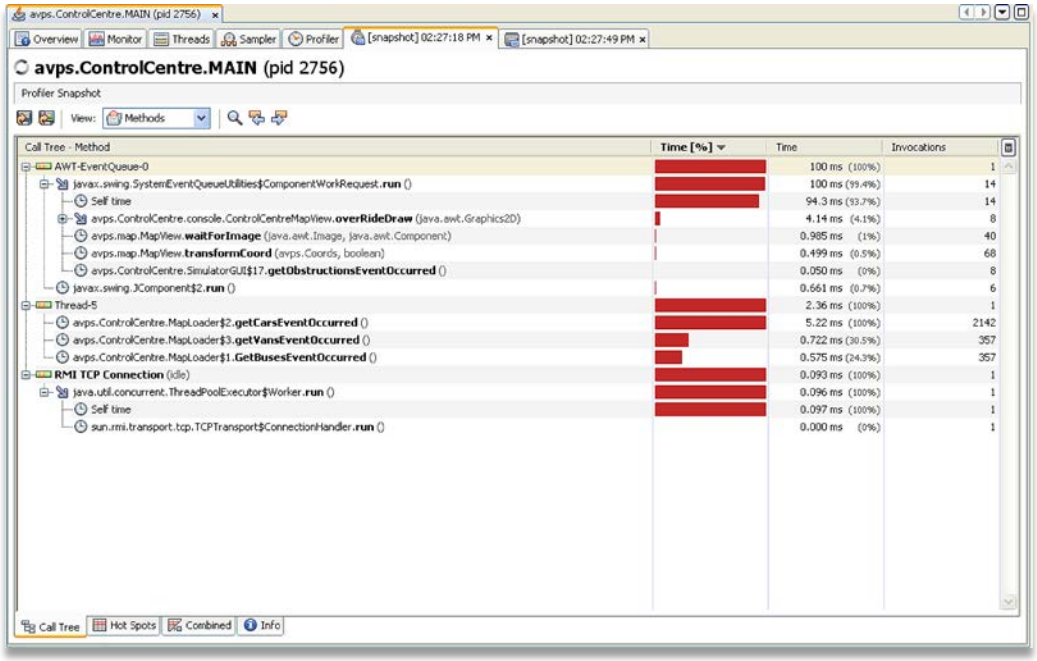


Fig. 6.37 Simulator II (PID 2756) profiling CPU

The memory profiles for the entering event, for Simulator I and II, are shown in Fig. 6.38 and Fig. 6.39. Simulator I consistently shows significantly higher memory

consumption for almost all objects compared to Simulator II. Some Simulator I objects such as int[] (33,216 bytes), have more double the memory consumption of their corresponding counterparts in Simulator II (16,472 bytes).

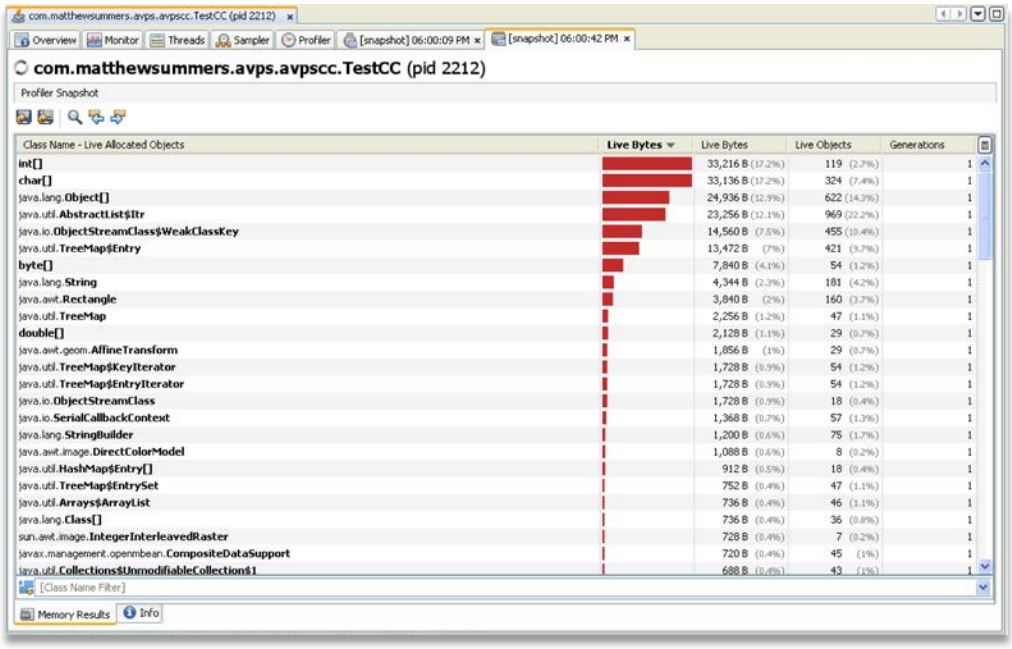


Fig. 6.38 Simulator I (PID 2212) profiling memory

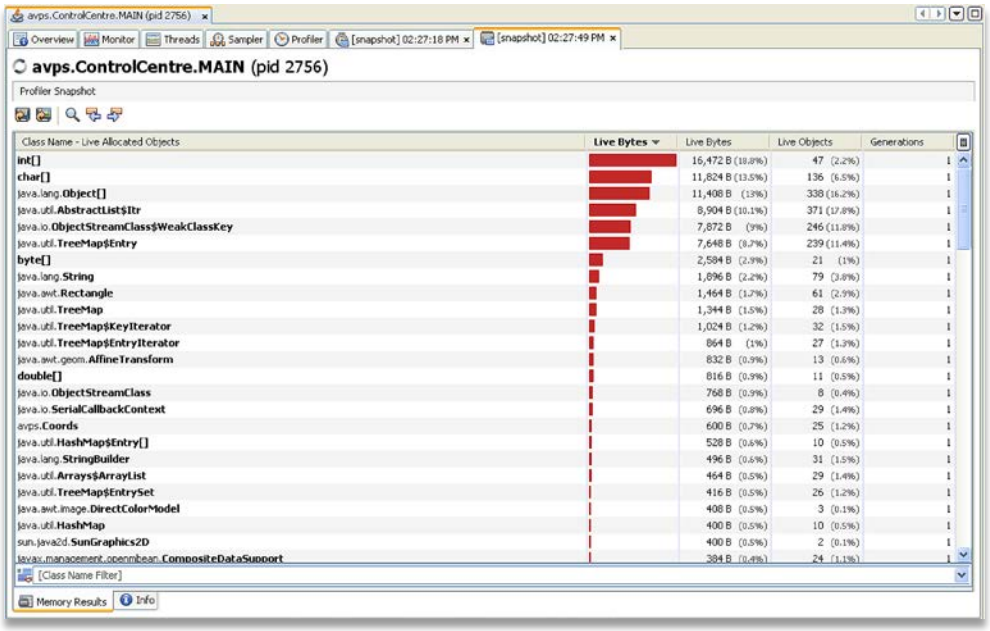


Fig. 6.39 Simulator II (PID 2756) profiling memory

Experiment Scenario 3: Two student vehicles, three visitor vehicles and two road closures

The objective of this experiment was to add complexity, in addition to increased load. The compares the behaviour of the two architectures a load of five vehicles; two student vehicles and three visitor vehicles, and two road closures. The requested destinations are the same as in Experiment Scenario 2. The student vehicles of type car are navigating to the Ash House, while two visitor vehicles are navigating to the Ruskin Library. One visitor vehicle is navigating to the Health Centre. However, there is the added complexity of two road closures (i.e. road id 17 and 38), which obstruct the direct route to Ruskin library (see Fig. 6.40 – 6.43). The closure has the effect of forcing the GVPS to compute a new shortest route to the Ruskin library.

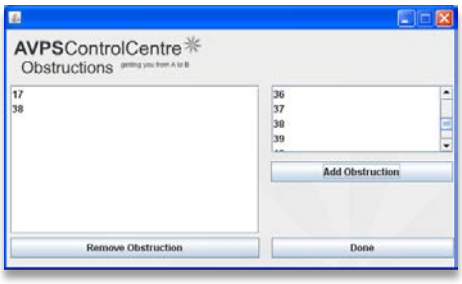


Fig. 6.40 Road obstruction menu

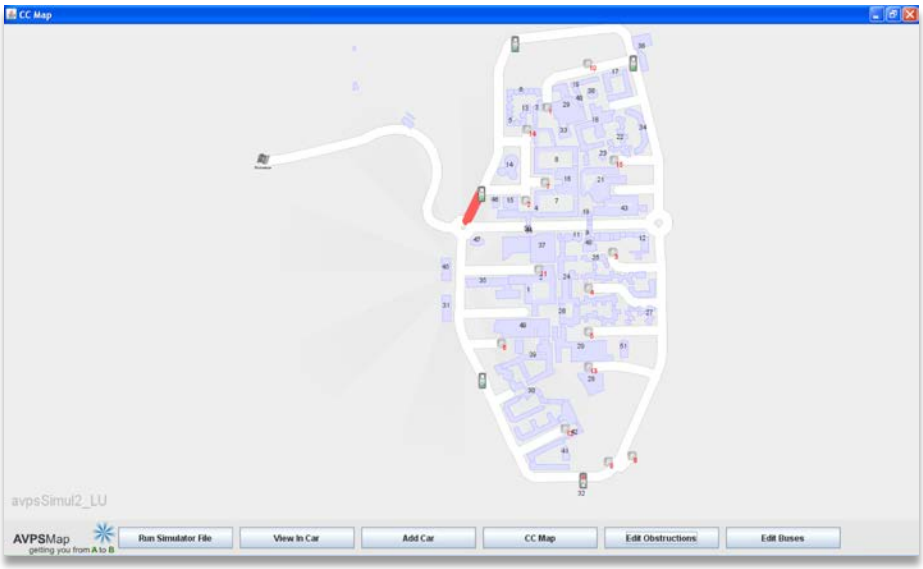


Fig. 6.41 Road obstruction at road 17 is shows on Control Centre panel of Simulator II

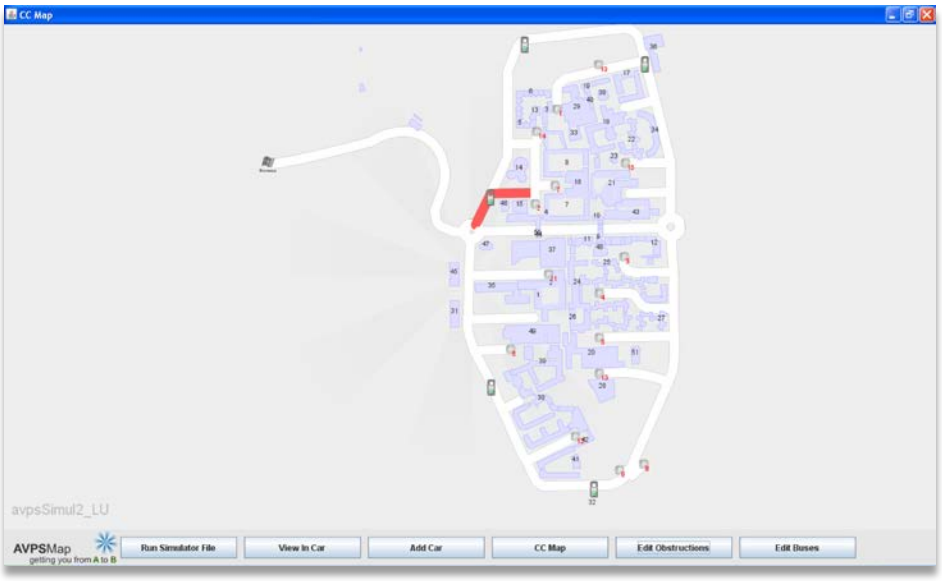


Fig. 6.42 Road obstruction at road 17 and 38 is shows on Control Centre panel of Simulator II

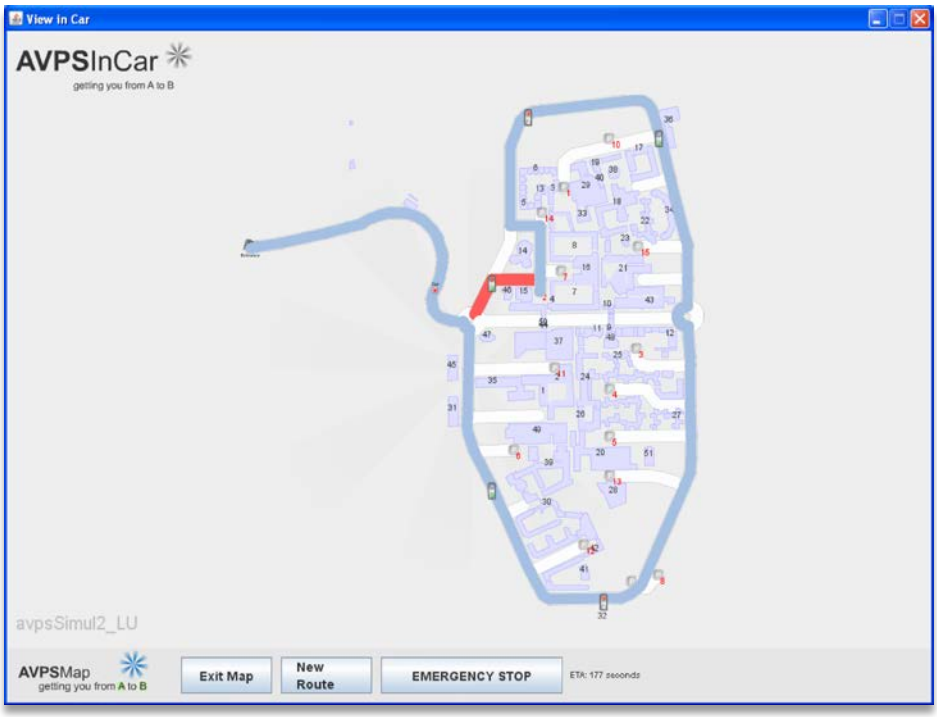


Fig. 6.43 Visitor car navigates to Ruskin Library using alternative road is shown in IVD panel of Simulator II

Performance and resource analysis for Experiment Scenario 3

The CPU, heap, classes and thread graphs for the entering event, for Simulator I and Simulator II, are shown in Fig. 6.44 and Fig. 6.45. For Simulator I, the event uses 1.6% of CPU time and 100,965,680 bytes of memory while loading 2,739 classes, and has a total of 28 live threads. Conversely, Simulator II uses 0.7% of CPU time and 94,998,656 bytes while loading 3,196 classes and 30 live threads. The memory consumption in the heap graphs show five spikes occurring after adding the same number of vehicles as in Experiment Scenario 2. Both simulators in Experiment Scenario 3 load more classes and threads than Experiment Scenario 1 and Experiment Scenario 2. Hence, the memory consumption in Experiment Scenario 3, for the entering event is the higher than in Experiment Scenario 1 and Experiment Scenario 2.

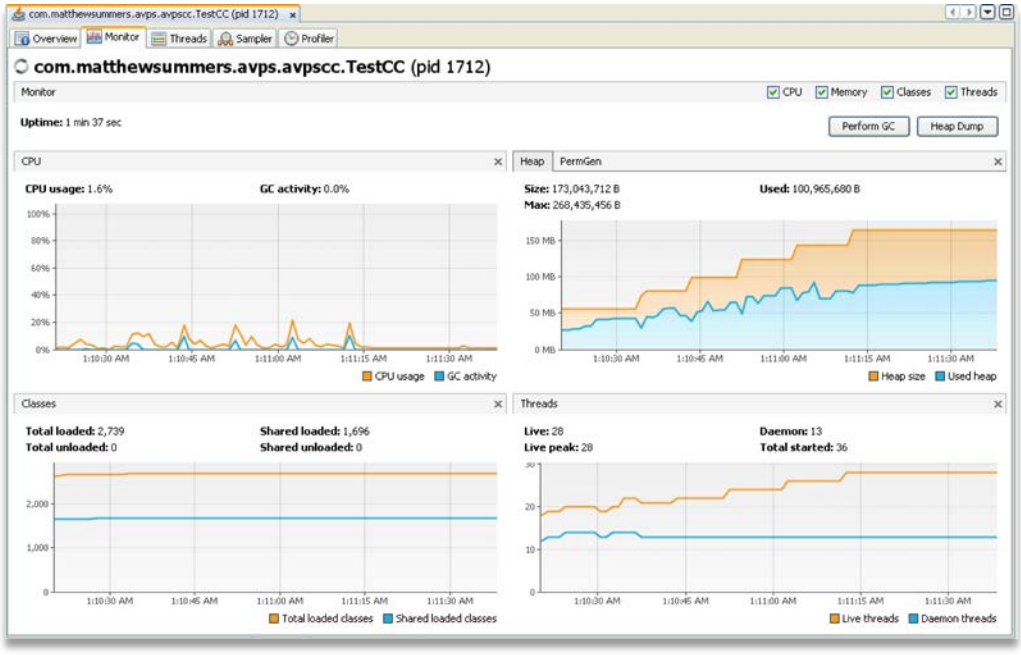


Fig. 6.44 Simulator I (PID 1712) monitoring entering event

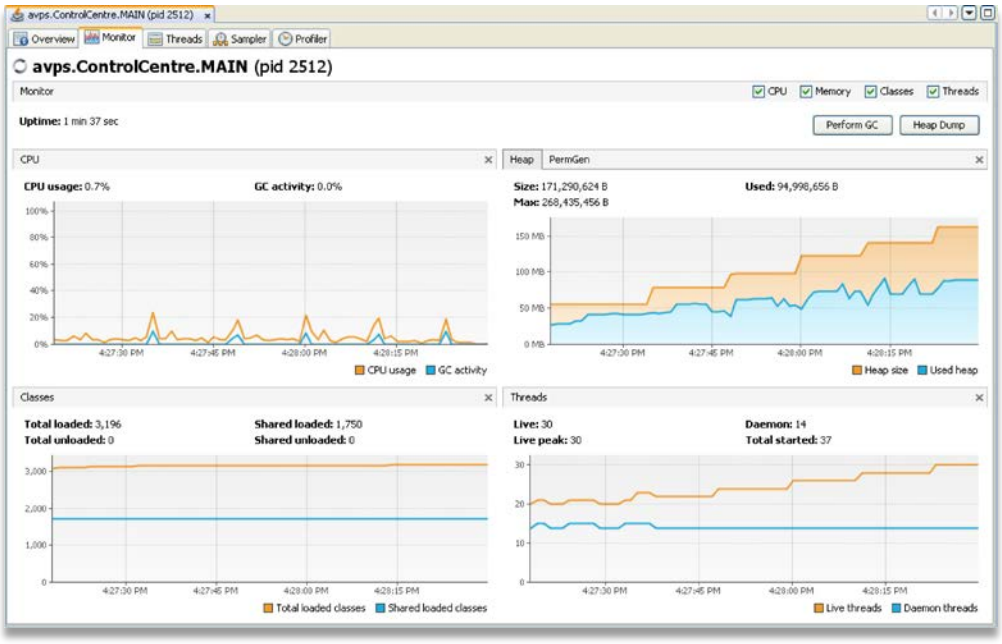


Fig. 6.45 Simulator II (PID 2512) monitoring entering event

The exiting event is triggered at approximately 750 seconds (see Fig. 6.46 and Fig. 6.47). The results show that Simulator I uses 2.4% of CPU time and 122,706,184 bytes of memory, which involves 4, 595 classes and 31 live threads. Simulator II uses 1.5% of CPU time and 119,025,464 bytes of memory while loading 5, 270 classes and 33 live threads. In the heap graphs, the memory consumption shows five spikes. These occur when the five vehicles exit their parking areas.

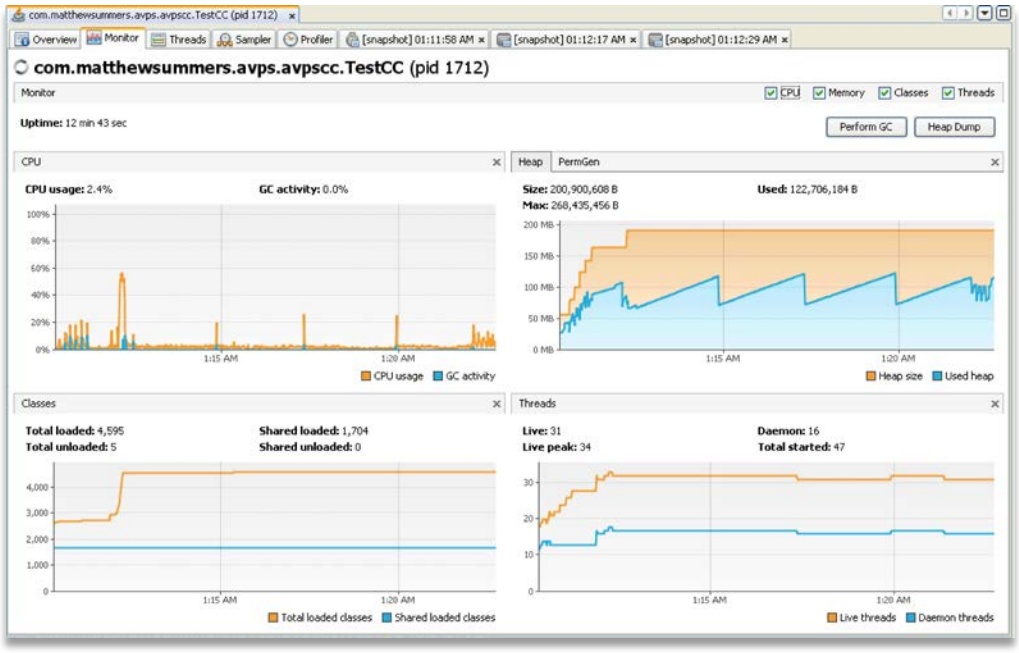


Fig. 6.46 Simulator I (PID 1712) monitoring exiting event

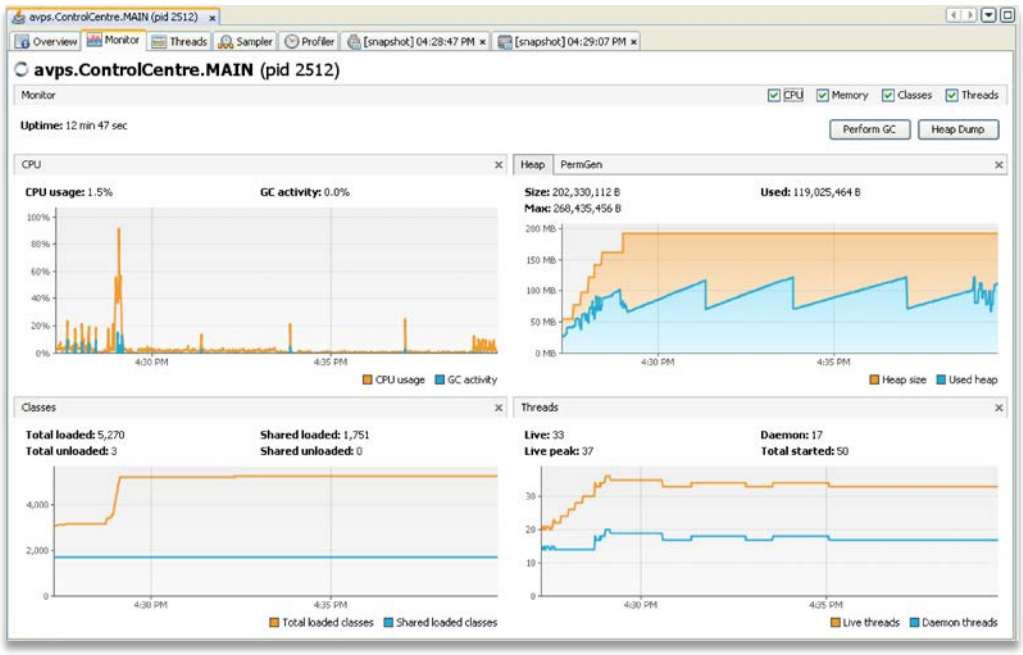


Fig. 6.47 Simulator II (PID 2512) monitoring exiting event

The results for Experiment Scenario 3 are summarised in Table 6.8. Once again the results show that Simulator II has performed significantly better than Simulator I

despite running more classes and threads. Simulator II has lower CPU usage and lower memory consumption compared to Simulator I.

Table 6.8 Summary performance and memory consumption for Experiment Scenario 3

		CPU Usage	Heap (bytes)	Classes	Threads
Entering event	Simulator I	1.6%	100, 965, 680	2, 739	28
	Simulator II	0.7%	94, 998, 656	3, 196	30
Exiting event	Simulator I	2.4%	122, 706, 184	4, 595	31
	Simulator II	1.5%	119, 025, 494	5, 270	33

CPU and memory profiles for Experiment Scenario 3

Fig. 6.48 and Fig. 6.49 show the CPU profile for Simulator I and Simulator II in Experiment Scenario 3 during entering event. Simulator I shows that the application spends almost all its time in the AWT-EventQueue-0 class. In Simulator II (Fig 6.49), the application spends most of its time in the Thread-5 and AWT-EventQueue-0 classes.

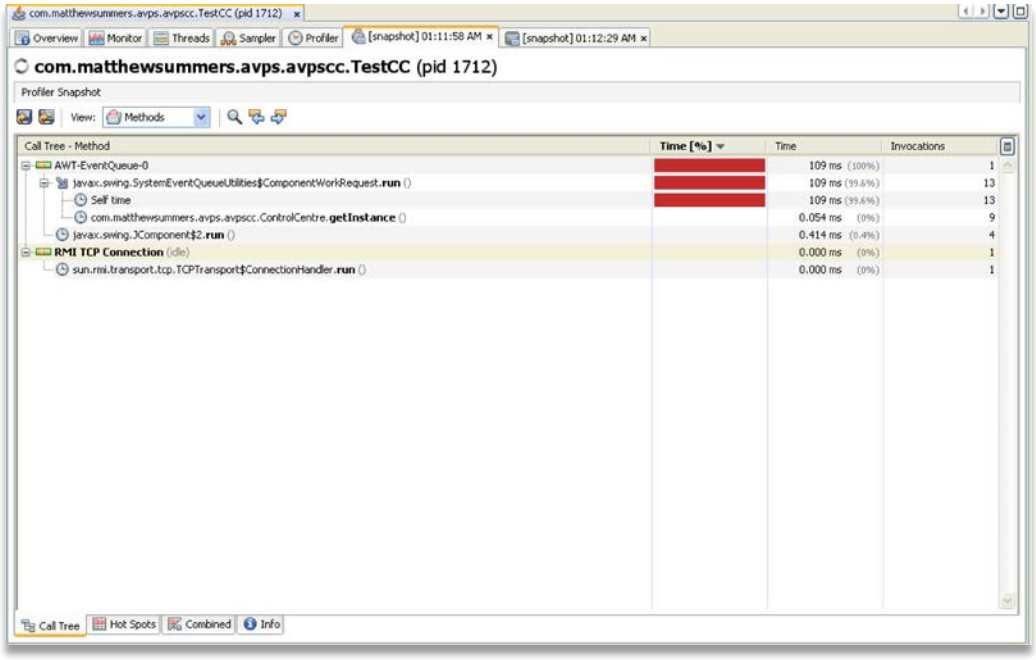


Fig. 6.48 Simulator I (PID 1712) profiling CPU

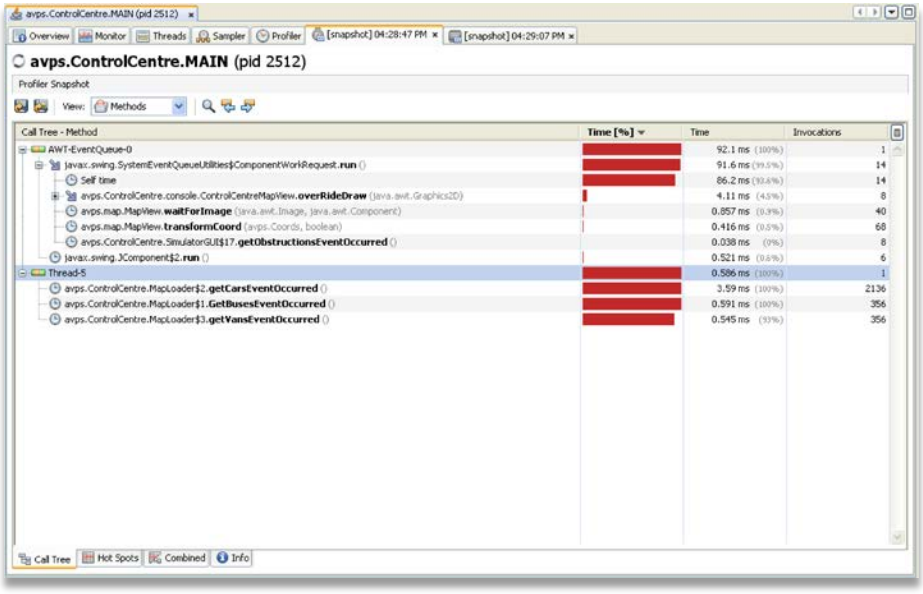


Fig. 6.49 Simulator II (PID 2512) profiling CPU

Memory profiles for the entering event for Simulator I and II in Experiment Scenario 3 are shown in Fig. 6.50 and Fig. 6.51. Simulator I shows significantly higher memory consumption for almost all objects compared to Simulator II. The object int[] of Simulator I, for example, has a memory consumption of bytes 58,128 bytes compared to Simulator II, where it's consumption is only 2,088 bytes.

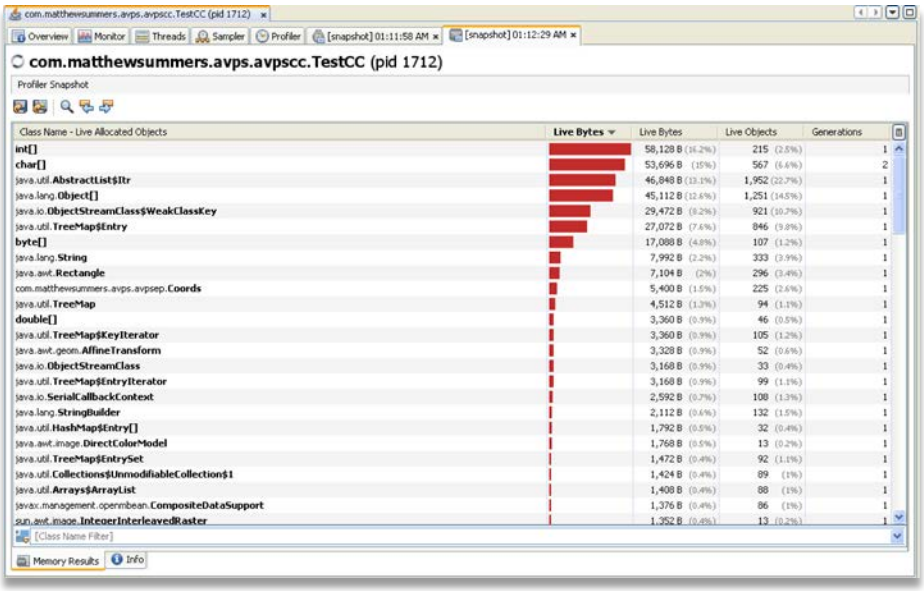


Fig. 6.50 Simulator II (PID 1712) profiling memory

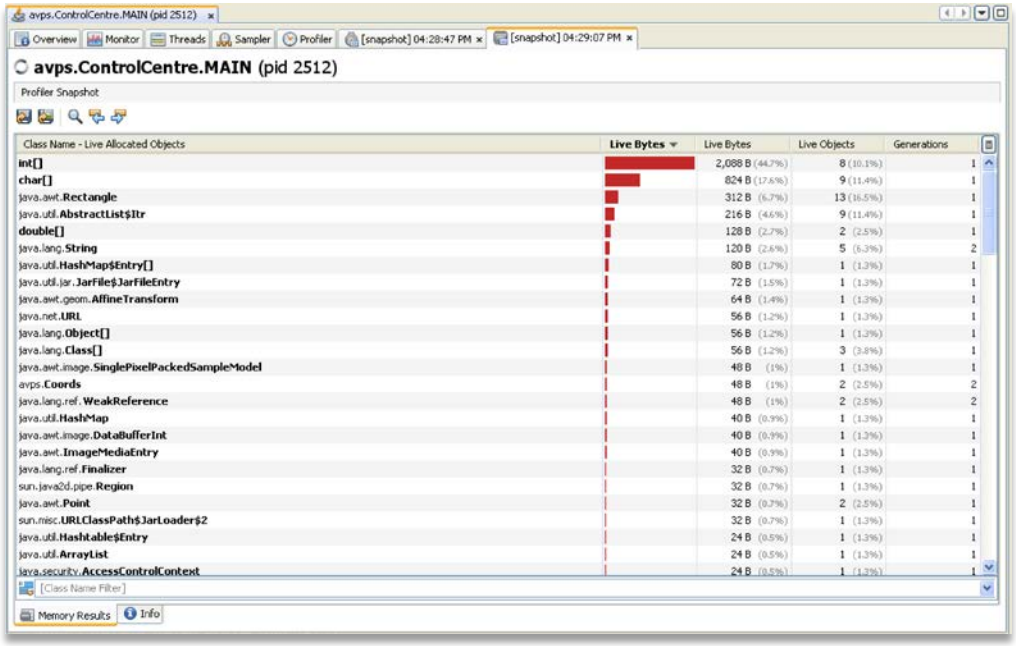


Fig. 6.51 Simulator II (PID 2512) profiling memory

6.5 Summary

This chapter has provided a runtime evaluation of CSAFE. The evaluation used a real case study derived from an undergraduate software engineering group project. The evaluation assessed the effect of CSAFE efficiency and performance refinements on the runtime behaviour of a system by comparing the original system with its refined version. In all cases the results validated the effectiveness of CSAFE by showing significant improvements in performance and resource consumption for the refined system.

Chapter 7

Conclusions

This chapter begins with an evaluation of the Component-based Software Architecture analysis FramEwork (CSAFE), a scenario-driven, negotiation-based architecture analysis approach that intended to provide a viable framework for architectural analysis in CBD. The evaluation compares the research achievements with the objectives outlined in the introduction chapter. The chapter then provides a discussion of future research directions, which have arisen during the development and evaluation of the framework. The chapter concludes the thesis with a summary of the research problems of developing an effective architecture analysis framework for CBD, the issues with existing research efforts, and the key contributions put forward in this thesis.

7.1 Framework Objectives Revisited

This section discusses how CSAFE has addressed the research objectives stated in the thesis introduction. These research objectives are to allow the system designer to adapt and tailor the design process to reflect the system context and domain specific needs, to provide support for pluggable architecture analysis, provide explicit support for

trade-off analysis, to provide support for standard design notations, and develop an extensible toolset to support the architecture analysis framework. How each objective has been addressed is now discussed:

1. *Formulate a classification and comparison framework for architecture analysis approaches.* This objective has been achieved by identifying the design challenges in component-based development and distilling them into a set of necessary requirements for architecture analysis methods. The set of necessary requirements has been used to develop a framework for assessing architecture analysis approaches, which in turn, has been successfully used to assess current architecture analysis methods [Admodisastro08].
2. *Develop a scenario-driven architecture analysis framework to support black-box component-based development.* This objective has been achieved by the development of Component–based Software Architecture analysis FramEwork (CSAFE). CSAFE competently addresses the design challenges outlined in Chapter 3 and has been successfully evaluated on both static and runtime case studies [Admodisastro10, Admodisastro11a, Admodisastro11b]. CSAFE supports the following features:
    - Is process-pluggable to minimise development process disruption and to afford system designers flexibility in the way they conduct architecture analysis to take into account application context needs. Output from the existing design process forms the input to the CSAFE process. The recommendations from architecture analysis process are fed back into the normal design process. This means that system designs do not have to modify their development process significantly to accommodate CSAFE.
    - Explicitly supports broad system stakeholder involvement in architecture analysis through analysis scenarios allow system stakeholders to tailor the analysis to explore specific design questions. CSAFE also maintains

traceability with the rest of the development process allowing for pluggable "what-if" analysis of design and evolution changes using analysis scenarios.

- Provides support for pluggable analysis to allow for diversity in analysis. Currently, the CSAFE analysis process illustrated with three types of checking, which include structure, behavioural, and conformance, for which different tools may be used.
- Supports for negotiation (i.e. trade-off analysis) is provided in the framework through the implementation of the *Simple Multi-Attribute Rating Technique (SMART)*, which is a form of the multi-attribute utility theory methods. Trade-off analysis supports the process of balancing stakeholder concerns and architectural considerations with the available component functionality.
- Supports architectures described in UML and those described in the iXML ADL. CSAFE incorporates a parser to translate UML architecture descriptions to iXML ADL, and a verifier for iXML architecture descriptions. The iXML ADL serves three purposes; first, it allows both pre-existing and new architectures to be analysed. Secondly, it allows for a portable, platform independent description of the system architecture. Lastly, it provides the system designer with a mechanism for augmenting architectural descriptions to explore "what if" analysis. The system architecture, architectural design templates and components specifications are all represented in the same way using a standard XML schema.
- Is primarily intended to support black-box component-based development. However, the approach recognises that there might be aspects of a system for which black-box development is not feasible or appropriate. In such cases, CSAFE supports custom development by treating abstract design components as placeholders for custom development. CSAFE supports hybrid component-based development in recognition that that component-based systems are increasingly hybrid

integrations of off-the-shelf components and web services. The evaluation in described Chapter 5 uses components and a web service.

3. *Develop an extensible toolset to support the architecture analysis framework.* This objective has been achieved through the development of an extensive CSAFE toolset that has six main components: The XMI/XML parser, scenario formulator, analyser, iXML ADL, trade-off analyser and rater, and report generator. These components are supported by an analysis repository containing the design template library, component library and architecture database. The primary aim of the toolset is to support the architecture analysis process. The CSAFE toolset achieves this by:
    - Providing explicit support for the involvement of system stakeholders and supporting the formulation analysis scenarios that explore specific design questions
    - Supporting the analysis of architectures specified in UML
    - Supporting diversity in analysis through pluggable analysis, to allow different forms of architecture analysis to be conducted
    - Supporting an extensible XML repository of design templates and components that allows the system designer to define analysis contexts that include design patterns, styles and organisation-specific schemes.
    - Providing explicit support for negotiation through the trade-off analyser and support assessment of proposed solutions

    The efficacy of the toolset is clearly demonstrated in the EDDIS case study described in Chapter 5 [Admodisastro10] and GVPS case study described in Chapter 6.

4. *Evaluate the architecture analysis framework on a non-trivial case study.* This objective has been achieved by clearly demonstrating the efficacy of CSAFE in two non-trivial design settings. Chapter 5 used the requirements of an actual Electronic Document Delivery and Interchange system (EDDIS) to demonstrate the key

features of CSAFE and the practicability of the framework. The evaluation demonstrated how CSAFE can be used to construct, analyse and refine a software system architecture from requirements to system composition. The second evaluation focused on runtime evaluation to validate architectural refinements. The evaluation assesses the effect of architectural refinements by comparing the runtime behaviour of an existing system against its refined version, in three different scenarios. In all cases the results validated the effectiveness of CSAFE by showing significant improvement in performance and resource consumption for the refined system.

7.3 Opportunities and Future Work

This section discusses ideas for future research stemming from the development and evaluation of the CSAFE. Each research direction is discussed in turn:

- *Improving service and component mapping process.* The service and component mapping process can be significantly improved semantically by the adoption of an ontology for describing properties and capabilities of services and concrete components. Current system services are mapped onto the recommended architecture design templates, and abstract components are mapped onto concrete components using semantic reasoning. An ontology can provide the semantic information for conducting more efficient mapping and analysis.
- *Enhancing negotiation support.* The framework supports negotiation using the SMART technique. CSAFE could explore other methods for conducting negotiation. For example, in addition to SMART, the Analytic Hierarchical Process technique (AHP) is also widely used in multi-attribute decision-making. The enhancement can be implemented in CSAFE as pluggable negotiation, allowing different trade-off analysis tools to be used. Different negotiation techniques vary in the wider decision factors that they consider. Supporting

flexibility in negotiation may allow aspects such as uncertainty and risk to be incorporated in the analysis.

- *Extending analysis process checkers.* Currently the CSAFE analysis process provides support for structure, quality and conformance checking. The output of the analysis process is a report outlining potential inconsistencies and mismatches, and recommendations for improving the architecture. The analysis could be extended to support behaviour checking by extending iXML ADL to support component behaviour specification. This would provide the basis dynamic analysis of architecture design.
- *Incorporating component metrics in assessment.* Component metrics provide useful quantitative information related to interface complexity, code size, component dependency and other measurable system attributes. There are numerous metrics available at the code level and some researchers also have worked on the design metrics. However, current literature shows there are few metrics at architectural level, and much less for black-box development. The quantitative evaluation using component metrics would be a useful addition to the decision-making process. Therefore, a study might be conducted to identify and incorporate useful component metrics to support the assessment process.
- *Dealing with large and complex system.* For CSAFE to be scalable it needs to demonstrate that it can cope with the analysis of large and complex systems. The prototype toolset has demonstrated that CSAFE can effectively analyse non-trivial system architectures. However, the evaluations described here represent only a small class of systems and a handful of scenarios. It is important that CSAFE is validated on larger, more complex applications, and more quality scenarios. Through further validations, the maturity of CSAFE will improve. Maturity indicates the state of readiness of CSAFE to be adopted in an organization.

7.4 Reflection

The importance of architecture in reuse-driven development is widely recognized [Bass05, Crnkovic02, Medvidovic07]. Architecture provides a framework for establishing a match between available components and the system context. It is a key part of the system documentation; it enforces the integrity of component composition and provides a basis for managing change. However, one of the most difficult problems in component-based system development (CBD) is ensuring that the software architecture provides an acceptable match with its intended application, business and evolutionary context. Unlike custom development where architectural design relies solely on detailed requirements specification and where deficiencies in application context can be corrected by 'tweaking' the source code, in component-based system development the typical unit of development is often a black-box component whose source code is inaccessible to the developer. Getting the architecture right is therefore key to ensuring quality in a component-based system.

In this thesis we highlighted how architecture analysis can provide the developer with a means to assess design configurations and to verify the adequacy of compositions with respect to stakeholder concerns. Architecture analysis can also provide a basis for developing "what-if" scenarios to explore the implications of evolving a system [Kotonya05a, Dobrica02]. However, a study by [Admodisastro08] showed that current architecture analysis approaches differ widely with respect to their underlying models and ability to support black-box software development making it difficult for system designer to assess their efficacy in different application contexts. The study also showed that there is significant disparity in the analytical capabilities and user validation of the approaches.

The need to trade-off and accept compromise is therefore central to the successful development of component-based systems. However, current architecture analysis approaches provide poor support for negotiation. This thesis has highlighted the poor

support for diversity in current architecture analysis approaches. Current approaches are largely designed to support a particular type of analysis and often for a specific application domain. Critically, none of the approaches reviewed in this thesis support hybrid reuse-driven development, even though, increasingly applications are being developed for which different types of reusable software co-exist in the same system. In current architecture analysis approaches, the role of architectural design is left largely to the system designer. However, system stakeholders often include decision makers within and outside the organisation and their involvement in architecture analysis can help identify critical system concerns and conflicts, assess alternatives and build consensus on priority issues.

This thesis has presented two key research contributions. The first key research contribution of this thesis is the formulation of a classification and comparison framework for software architecture analysis approaches. The framework consists of eight key requirements that can be used to design architectural methods and assess efficacy for component-based development. The second key contribution is development and evaluation of Component-based Software Architectural Analysis Framework (CSAFE), a scenario driven, negotiation-based architecture analysis framework for black-box component-based software development. It is important to mention that while CSAFE is primarily intended to support black-box development, we recognise that there may be aspects of the system for which a black-box solution is not feasible. CSAFE supports white-box development in such situations by treating abstract components as placeholders for custom development.

This thesis has highlighted the importance of architectural analysis in component-based software development. Systematic architectural analysis can help ensure that risks resulting from architectural adaptations and trade-offs do not adversely affect critical system qualities. The analysis is likely to reveal not only how well an architecture satisfies a particular application context, but also how change to specific quality attributes might affect other quality concerns. The work does not pretend that it

has addresses all the problems posed by black-box component-based system design. However, it is believe that the work has made significant contribution in understanding and addressing those problems.

Appendix A:

iXML Schemas

A1. iXML Schema for Architecture Design Description

iXML schema for architecture design is shown in Table A1.1.

Table A1.1 iXML schema for architecture

```
<!ELEMENT NXML (COMPONENT*, INTERFACE*, CONNECTOR*)>

<!ELEMENT COMPONENT (COMPONENT.DESCRIPTION, COMPONENT.INTERFACE*,
COMPONENT.CONNECTOR*, COMPONENT.COMPOSITE*, COMPONENT.CONSTRAINT*,
COMPONENT.PROPERTY*)>
<!ATTLIST COMPONENT NAME.ID CDATA #REQUIRED
                    TYPE CDATA #IMPLIED
                    VISIBILITY CDATA #REQUIRED>

<!ELEMENT COMPONENT.DESCRIPTION (#PCDATA)>

<!-- INTERFACE PORT: P (PROVIDED) OR R (REQUIRED) INTERFACE -->
<!ELEMENT COMPONENT.INTERFACE EMPTY>
<!ATTLIST COMPONENT.INTERFACE NAME.IDREF CDATA #REQUIRED
                              PORT.IDREF CDATA #REQUIRED>

<!--LIST CONNECTOR ONLY FROM REQUIRE COMPONENT -->
<!ELEMENT COMPONENT.CONNECTOR EMPTY>
<!ATTLIST COMPONENT.CONNECTOR NAME.IDREF CDATA #REQUIRED>
```

```
<!--LIST COMPOSITE FOR NESTED COMPONENT OR SUBSYSTEM -->
<!ELEMENT COMPONENT.COMPOSITE EMPTY>
<!ATTLIST COMPONENT.COMPOSITE NAME.IDREF CDATA #REQUIRED>

<!-- CONSTRAINT STATE: NE, LE, LT, GE, GT OR EL -->
<!-- CONSTRAINT TYPE: PRECONDITION, POSTCONDITION OR INVARIANT -->
<!ELEMENT COMPONENT.CONSTRAINT EMPTY>
<!ATTLIST COMPONENT.CONSTRAINT CONCERN CDATA #REQUIRED
                                SUBCONCERN CDATA #REQUIRED
                                STATE CDATA #REQUIRED
                                TYPE CDATA #REQUIRED
                                VALUE CDATA #REQUIRED
                                SCOPE CDATA #REQUIRED>

<!ELEMENT COMPONENT.PROPERTY EMPTY>
<!ATTLIST COMPONENT.PROPERTY CONCERN CDATA #REQUIRED
                              SUBCONCERN CDATA #REQUIRED
                              VALUE CDATA #REQUIRED>

<!ELEMENT INTERFACE (INTERFACE.DESCRIPTION, INTERFACE.SERVICE*, INTERFACE.OPERATION*,
INTERFACE.CONSTRAINT*, INTERFACE.PROPERTY*)>
<!ATTLIST INTERFACE NAME.ID CDATA #REQUIRED
                    TYPE CDATA #IMPLIED
                    PORT CDATA #IMPLIED
                    VISIBILITY CDATA #REQUIRED >

<!ELEMENT INTERFACE.DESCRIPTION (#PCDATA)>

<!--LIST SERVICE FOR PROVIDED INTERFACE -->
<!ELEMENT INTERFACE.SERVICE EMPTY>
<!ATTLIST INTERFACE.SERVICE NAME CDATA #IMPLIED>

<!ELEMENT INTERFACE.OPERATION (OPERATION.PARAM*)>
<!ATTLIST INTERFACE.OPERATION NAME CDATA #IMPLIED
                RET CDATA #IMPLIED>

<!ELEMENT OPERATION.PARAM EMPTY>
<!ATTLIST OPERATION.PARAM NAME CDATA #IMPLIED
                          TYPE CDATA #IMPLIED>

<!ELEMENT INTERFACE.CONSTRAINT EMPTY>
<!ATTLIST INTERFACE.CONSTRAINT CONCERN CDATA #REQUIRED
                               SUBCONCERN CDATA #REQUIRED
                               TYPE CDATA #REQUIRED
                               STATE CDATA #REQUIRED
                               VALUE CDATA #REQUIRED
                               SCOPE CDATA #REQUIRED>

<!ELEMENT INTERFACE.PROPERTY EMPTY>
<!ATTLIST INTERFACE.PROPERTY CONCERN CDATA #REQUIRED
                             SUBCONCERN CDATA #REQUIRED
                             VALUE CDATA #REQUIRED>

<!ELEMENT CONNECTOR (CONNECTOR.REQUIRED, CONNECTOR.PROVIDED, CONNECTOR.CONSTRAINT*,
CONNECTOR.PROPERTY*)>
```

```
<!ATTLIST CONNECTOR NAME.ID CDATA #IMPLIED
                    TYPE CDATA #IMPLIED
                    ROLE CDATA #IMPLIED>

<!ELEMENT CONNECTOR.PROVIDED (PROVIDED.COMPONENT, PROVIDED.INTERFACE)>
<!ELEMENT CONNECTOR.REQUIRED (REQUIRED.COMPONENT, REQUIRED.INTERFACE)>

<!ELEMENT PROVIDED.COMPONENT EMPTY>
<!ATTLIST PROVIDED.COMPONENT NAME.IDREF CDATA #REQUIRED>

<!ELEMENT PROVIDED.INTERFACE EMPTY>
<!ATTLIST PROVIDED.INTERFACE NAME.IDREF CDATA #REQUIRED>

<!ELEMENT REQUIRED.COMPONENT EMPTY>
<!ATTLIST REQUIRED.COMPONENT NAME.IDREF CDATA #REQUIRED>

<!ELEMENT REQUIRED.INTERFACE EMPTY>
<!ATTLIST REQUIRED.INTERFACE NAME.IDREF CDATA #REQUIRED>

<!ELEMENT CONNECTOR.CONSTRAINT EMPTY>
<!ATTLIST CONNECTOR.CONSTRAINT CONCERN CDATA #REQUIRED
                               SUBCONCERN CDATA #REQUIRED
                               STATE CDATA #REQUIRED
                               TYPE CDATA #REQUIRED
                               VALUE CDATA #REQUIRED
                               SCOPE CDATA #REQUIRED>

<!ELEMENT CONNECTOR.PROPERTY EMPTY>
<!ATTLIST CONNECTOR.PROPERTY CONCERN CDATA #REQUIRED
                             SUBCONCERN CDATA #REQUIRED
                             VALUE CDATA #REQUIRED>
```

A2. iXML Schema for Design Template Description

iXML schema for design template is shown in Table A2.1.

Table A2.1 iXML schema for design template

```
<!ELEMENT NXML (CATEGORY, RNAME, ALSOKNOWNAS, RELATEDRULES, INTENT, CONTEXT,
MOTIVATION, CONTRIBUTIONS, CONFIGURATION)>

<!--CATEGORY: PATTERN, STYLE, LOCAL SCHEME -->
<!ELEMENT CATEGORY (#PCDATA)>
<!ELEMENT RNAME (#PCDATA)>
<!ELEMENT ALSOKNOWNAS (#PCDATA)>
<!ELEMENT RELATEDRULES (RELATEDRULE.DESCRIPTION*)>

<!ELEMENT RELATEDRULE.DESCRIPTION EMPTY>
<!ATTLIST RELATEDRULE.DESCRIPTION RNAME CDATA #REQUIRED >

<!ELEMENT INTENT (#PCDATA)>
<!ELEMENT CONTEXT (#PCDATA)>
<!ELEMENT MOTIVATION (#PCDATA)>
<!ELEMENT CONTRIBUTIONS (CONTRIBUTION.DESCRIPTION*)>

<!-- CONSEQUENCES WEIGHT: H(HIGH), M(MEDIUM) OR L(LOW) -->
<!ELEMENT CONTRIBUTION.DESCRIPTION (#PCDATA)>
<!ATTLIST CONTRIBUTION.DESCRIPTION QUALITY CDATA #REQUIRED
                                   SUBQUALITY CDATA #REQUIRED
                                   WEIGHT CDATA #REQUIRED>

<!ELEMENT CONFIGURATION (COMPONENT*, INTERFACE*, CONNECTOR*)>

<!ELEMENT COMPONENT (COMPONENT.DESCRIPTION, COMPONENT.INTERFACE*,
COMPONENT.CONNECTOR*, COMPONENT.COMPOSITE*, COMPONENT.CONSTRAINT*,
COMPONENT.PROPERTY*)>
<!ATTLIST COMPONENT NAME.ID CDATA #REQUIRED
                    TYPE CDATA #IMPLIED
                    VISIBILITY CDATA #REQUIRED >

<!ELEMENT COMPONENT.DESCRIPTION (#PCDATA)>

<!-- INTERFACE PORT: P (PROVIDED) OR R (REQUIRED) INTERFACE -->
<!ELEMENT COMPONENT.INTERFACE EMPTY>
<!ATTLIST COMPONENT.INTERFACE NAME.IDREF CDATA #REQUIRED
                              PORT.IDREF CDATA #REQUIRED>

<!--LIST CONNECTOR ONLY FROM REQUIRE COMPONENT -->
<!ELEMENT COMPONENT.CONNECTOR EMPTY>
<!ATTLIST COMPONENT.CONNECTOR NAME.IDREF CDATA #REQUIRED>

<!--LIST COMPOSITE FOR NESTED COMPONENT OR SUBSYSTEM -->
<!ELEMENT COMPONENT.COMPOSITE EMPTY>
<!ATTLIST COMPONENT.COMPOSITE NAME.IDREF CDATA #REQUIRED>
```

```
<!-- CONSTRAINT STATE: NE, LE, LT, GE, GT OR EL -->
<!-- CONSTRAINT TYPE: PRECONDITION, POSTCONDITION OR INVARIANT -->
<!ELEMENT COMPONENT.CONSTRAINT EMPTY>
<!ATTLIST COMPONENT.CONSTRAINT CONCERN CDATA #REQUIRED
                               SUBCONCERN CDATA #REQUIRED
                               STATE CDATA #REQUIRED
                               TYPE CDATA #REQUIRED
                               VALUE CDATA #REQUIRED
                               SCOPE CDATA #REQUIRED>

<!ELEMENT COMPONENT.PROPERTY EMPTY>
<!ATTLIST COMPONENT.PROPERTY CONCERN CDATA #REQUIRED
                             SUBCONCERN CDATA #REQUIRED
                             VALUE CDATA #REQUIRED>

<!ELEMENT INTERFACE (INTERFACE.DESCRIPTION, INTERFACE.SERVICE*, INTERFACE.OPERATION*,
INTERFACE.CONSTRAINT*, INTERFACE.PROPERTY*)>
<!ATTLIST INTERFACE NAME.ID CDATA #REQUIRED
                    TYPE CDATA #IMPLIED
                    PORT CDATA #IMPLIED
                    VISIBILITY CDATA #REQUIRED >

<!ELEMENT INTERFACE.DESCRIPTION (#PCDATA)>

<!--LIST SERVICE FOR PROVIDED INTERFACE -->
<!ELEMENT INTERFACE.SERVICE EMPTY>
<!ATTLIST INTERFACE.SERVICE NAME CDATA #IMPLIED>

<!ELEMENT INTERFACE.OPERATION (OPERATION.PARAM*)>
<!ATTLIST INTERFACE.OPERATION NAME CDATA #IMPLIED
                              RET CDATA #IMPLIED>

<!ELEMENT OPERATION.PARAM EMPTY>
<!ATTLIST OPERATION.PARAM NAME CDATA #IMPLIED
                          TYPE CDATA #IMPLIED>

<!ELEMENT INTERFACE.CONSTRAINT EMPTY>
<!ATTLIST INTERFACE.CONSTRAINT CONCERN CDATA #REQUIRED
                               SUBCONCERN CDATA #REQUIRED
                               TYPE CDATA #REQUIRED
                               STATE CDATA #REQUIRED
                               VALUE CDATA #REQUIRED
                               SCOPE CDATA #REQUIRED>

<!ELEMENT INTERFACE.PROPERTY EMPTY>
<!ATTLIST INTERFACE.PROPERTY CONCERN CDATA #REQUIRED
                             SUBCONCERN CDATA #REQUIRED
                             VALUE CDATA #REQUIRED>

<!ELEMENT CONNECTOR (CONNECTOR.REQUIRED, CONNECTOR.PROVIDED, CONNECTOR.CONSTRAINT*,
CONNECTOR.PROPERTY*)>
<!ATTLIST CONNECTOR NAME.ID CDATA #IMPLIED
                    TYPE CDATA #IMPLIED
                    ROLE CDATA #IMPLIED>
```

```
<!ELEMENT CONNECTOR.PROVIDED (PROVIDED.COMPONENT, PROVIDED.INTERFACE)>
<!ELEMENT CONNECTOR.REQUIRED (REQUIRED.COMPONENT, REQUIRED.INTERFACE)>

<!ELEMENT PROVIDED.COMPONENT EMPTY>
<!ATTLIST PROVIDED.COMPONENT NAME.IDREF CDATA #REQUIRED>

<!ELEMENT PROVIDED.INTERFACE EMPTY>
<!ATTLIST PROVIDED.INTERFACE NAME.IDREF CDATA #REQUIRED>

<!ELEMENT REQUIRED.COMPONENT EMPTY>
<!ATTLIST REQUIRED.COMPONENT NAME.IDREF CDATA #REQUIRED>

<!ELEMENT REQUIRED.INTERFACE EMPTY>
<!ATTLIST REQUIRED.INTERFACE NAME.IDREF CDATA #REQUIRED>

<!ELEMENT CONNECTOR.CONSTRAINT EMPTY>
<!ATTLIST CONNECTOR.CONSTRAINT CONCERN CDATA #REQUIRED
                               SUBCONCERN CDATA #REQUIRED
                               STATE CDATA #REQUIRED
                               TYPE CDATA #REQUIRED
                               VALUE CDATA #REQUIRED
                               SCOPE CDATA #REQUIRED>

<!ELEMENT CONNECTOR.PROPERTY EMPTY>
<!ATTLIST CONNECTOR.PROPERTY CONCERN CDATA #REQUIRED
                             SUBCONCERN CDATA #REQUIRED
                             VALUE CDATA #REQUIRED>
```

A3. iXML Schema for Component Description

iXML schema for component is shown in Table A3.1.

Table A3.1 iXML schema for component

```
<!ELEMENT NXML (COMPONENT*, INTERFACE*)>

<!ELEMENT COMPONENT (COMPONENT.DESCRIPTION, COMPONENT.INTERFACE*,
COMPONENT.CONSTRAINT*, COMPONENT.PROPERTY*)>
<!ATTLIST COMPONENT NAME.ID CDATA #REQUIRED
                    TYPE CDATA #IMPLIED
                    VISIBILITY CDATA #REQUIRED>

<!ELEMENT COMPONENT.DESCRIPTION (#PCDATA)>

<!-- INTERFACE PORT: P (PROVIDED) OR R (REQUIRED) INTERFACE -->
<!ELEMENT COMPONENT.INTERFACE EMPTY>
<!ATTLIST COMPONENT.INTERFACE NAME.IDREF CDATA #REQUIRED
                              PORT.IDREF CDATA #REQUIRED>

<!-- CONSTRAINT STATE: NE, LE, LT, GE, GT OR EL -->
<!-- CONSTRAINT TYPE: PRECONDITION, POSTCONDITION OR INVARIANT -->
<!ELEMENT COMPONENT.CONSTRAINT EMPTY>
<!ATTLIST COMPONENT.CONSTRAINT CONCERN CDATA #REQUIRED
                               SUBCONCERN CDATA #REQUIRED
                               STATE CDATA #REQUIRED
                               TYPE CDATA #REQUIRED
                               VALUE CDATA #REQUIRED
                               SCOPE CDATA #REQUIRED>

<!ELEMENT COMPONENT.PROPERTY EMPTY>
<!ATTLIST COMPONENT.PROPERTY CONCERN CDATA #REQUIRED
                             SUBCONCERN CDATA #REQUIRED
                             VALUE CDATA #REQUIRED>

<!ELEMENT INTERFACE (INTERFACE.DESCRIPTION, INTERFACE.SERVICE*, INTERFACE.OPERATION*,
INTERFACE.CONSTRAINT*, INTERFACE.PROPERTY*)>
<!ATTLIST INTERFACE NAME.ID CDATA #REQUIRED
                    TYPE CDATA #IMPLIED
                    PORT CDATA #IMPLIED
                    VISIBILITY CDATA #REQUIRED >

<!ELEMENT INTERFACE.DESCRIPTION (#PCDATA)>

<!--LIST SERVICE FOR PROVIDED INTERFACE -->
<!ELEMENT INTERFACE.SERVICE EMPTY>
<!ATTLIST INTERFACE.SERVICE NAME CDATA #IMPLIED>

<!ELEMENT INTERFACE.OPERATION (OPERATION.PARAM*)>
<!ATTLIST INTERFACE.OPERATION NAME CDATA #IMPLIED
                  RET CDATA #IMPLIED>
```

```
<!ELEMENT OPERATION.PARAM EMPTY>
<!ATTLIST OPERATION.PARAM NAME CDATA #IMPLIED
                          TYPE CDATA #IMPLIED>

<!ELEMENT INTERFACE.CONSTRAINT EMPTY>
<!ATTLIST INTERFACE.CONSTRAINT CONCERN CDATA #REQUIRED
                               SUBCONCERN CDATA #REQUIRED
                               TYPE CDATA #REQUIRED
                               STATE CDATA #REQUIRED
                               VALUE CDATA #REQUIRED
                               SCOPE CDATA #REQUIRED>

<!ELEMENT INTERFACE.PROPERTY EMPTY>
<!ATTLIST INTERFACE.PROPERTY CONCERN CDATA #REQUIRED
                             SUBCONCERN CDATA #REQUIRED
                             VALUE CDATA #REQUIRED>
```

Appendix B:

CSAFE Toolset Analysis & Design

B1. CSAFE Use-Case Descriptions & Sequence Diagrams

CSAFE use-case descriptions are shown in Table B1.1 – B1.13 and sequence diagrams are shown in Fig. B1.1 – B1.13.

Table B1.1 Transform architecture use-case description

CSAFE: Transform Architecture	
Actors	System Designer, XMI/XML Parser, Analysis Repository
Description	1. System designer browses and selects XMI/XML architectural specification from Analysis Repository. 2. System designer enters project name and clicks OK. 3. The XMI/XML parser parses the architectural specification and checks against XML schema/DTD. 4. The XMI/XML parser creates design schema. 5. The XMI/XML parser stores architectural vectors in analysis repository. 6. The tool organizes architectural elements in tree hierarchy and each element detail description is display on the description form.
Data	XMI/XML architectural specification
Stimulus	System designer selects 'New Project' from CSAFE File menu
Response	CSAFE parses and stores architecture design in to analysis repository.
Alternative flow of events	3.a. Invalid XMI/XML description. Indicate error message.

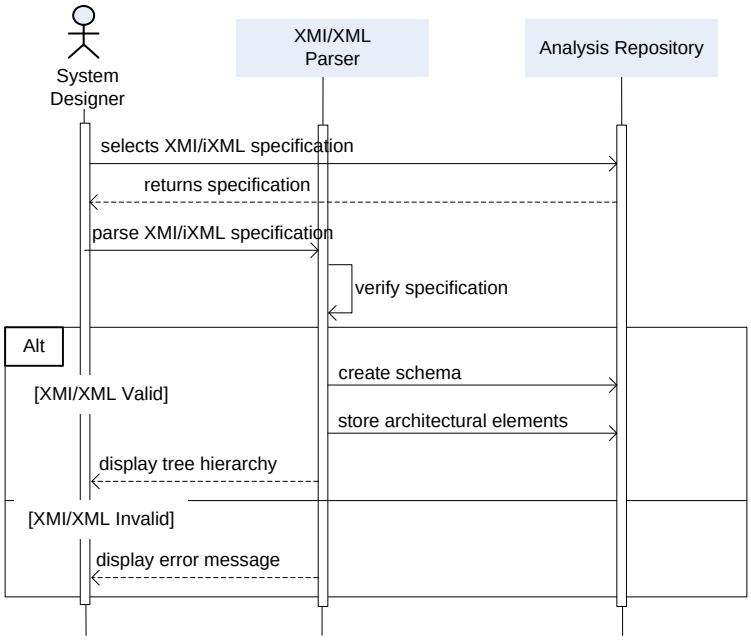


Fig. B1.1 Transform architecture sequence diagram

Table B1.2 Formulate scenario use-case description

CSAFE: Formulate Scenario	
Actors	System Designer, Analysis Repository
Description	1. System designer enters scenario descriptions (i.e. name, author and comment). A date and time automatically captures. 2. System designer clicks 'OK'. 3. The tool create new scenario in the analysis repository and display a new a scenario template on Elicit & Prioritise display. 4. System designer selects a node (composite component, service, interface or connector) from tree project and the constraint description is shows on the Elicit & Prioritise display. 5. Then the system designer can starts to weight each of the constraint description. 6. The system designer clicks 'Save' and the weighting values are store in Analysis Repository.
Data	New scenario and weighting values
Stimulus	System designer selects 'New Scenario' from CSAFE toolbar
Response	CSAFE stores new project info. and constraint's weighting values in to project repository.
Alternative flow of events	1.a. Duplicate scenario name. Indicate error message.

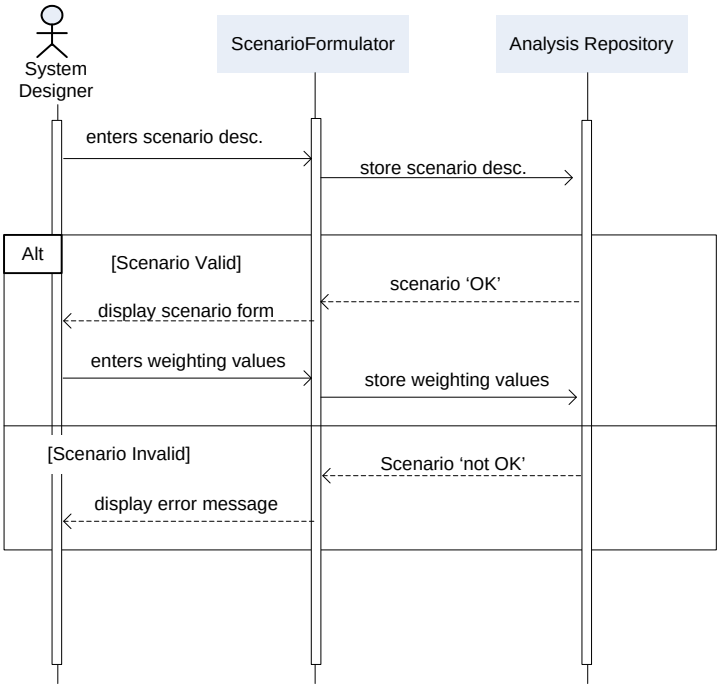


Fig. B1.2 Formulate scenario sequence diagram

Table B1.3 Analyse architecture use-case description

CSAFE: Analyse Architecture	
Actor	System Designer, Analysis Repository
Description	1. System designer executes mapping design use-case. 2. System designer selects conformance checker and retrieves analysis data from Analysis Repository. 3. The tool executes rating design use-case. 4. System designer selects quality checker and retrieves analysis data from Analysis Repository. 5. The tool executes mapping services use-cases and executes mapping component use-cases 6. System designer selects structural checker and retrieves analysis data from Analysis Repository.
Data	Formulates scenarios
Stimulus	System designer selects mapping form
Response	Results of conformance, quality and structural checker.
Relationship Extend:	Mapping design, Rating design, Mapping services, Mapping component
Alternative flow of events	-

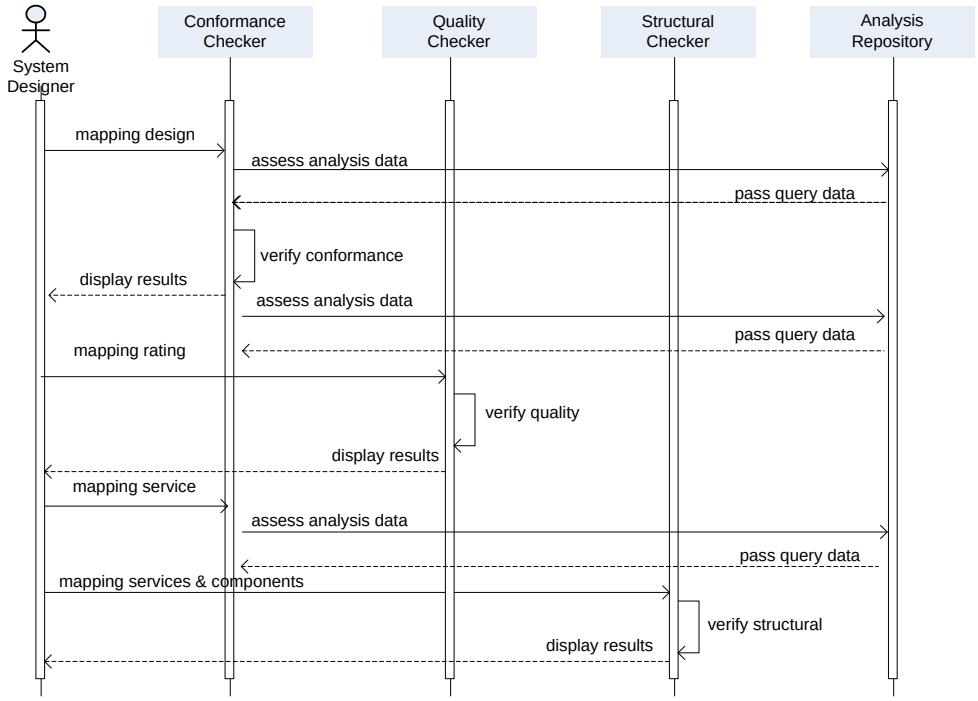


Fig. B1.3 Analyse architecture sequence diagram

Table B1.4 Map design use-case description

CSAFE: Map Design	
Actor	System designer, Analysis Repository
Description	1. System designer selects scenario name, quality concerns and design template categories. 2. The system design form submits this request to design control which then queries desired concerns and matching categories from design template repository. 3. The query results are passes to design control which then conduct comparison and matching. 4. The matching results are store in analysis repository and a success message is display to the designer.
Data	Quality concerns and design template category (i.e. Pattern, Local or Style)
Stimulus	System designer selects design mapping template.
Response	CSAFE stores design templates results in to project repository.
Alternative flow of events	3.a. Matching design template not found. Indicate error message.

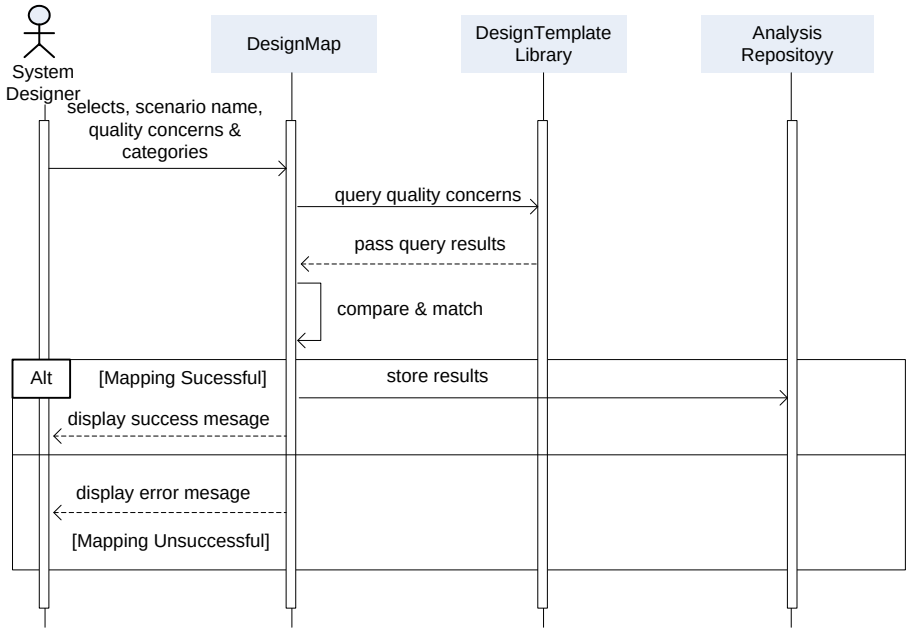


Fig. B1.4 Map design sequence diagram

Table B1.5 Rate design use-case description

CSAFE: Rate Design	
Actor	System Designer, Analysis Repository
Description	1. System designer selects scenario name, rating control submits the requests to analysis repository and retrieves mapping results. 2. Rating control then retrieves the design contributions from design template repository and passes the results to rating form to display rating for each design template. 3. Then the system designer instantiated desired alternatives designs and its justifications. 4. These architectural instantiation are store in architecture database.
Data	All related design templates.
Stimulus	System designer selects rate map template.
Response	CSAFE stores desired design template and its justification.
Alternative flow of events	-

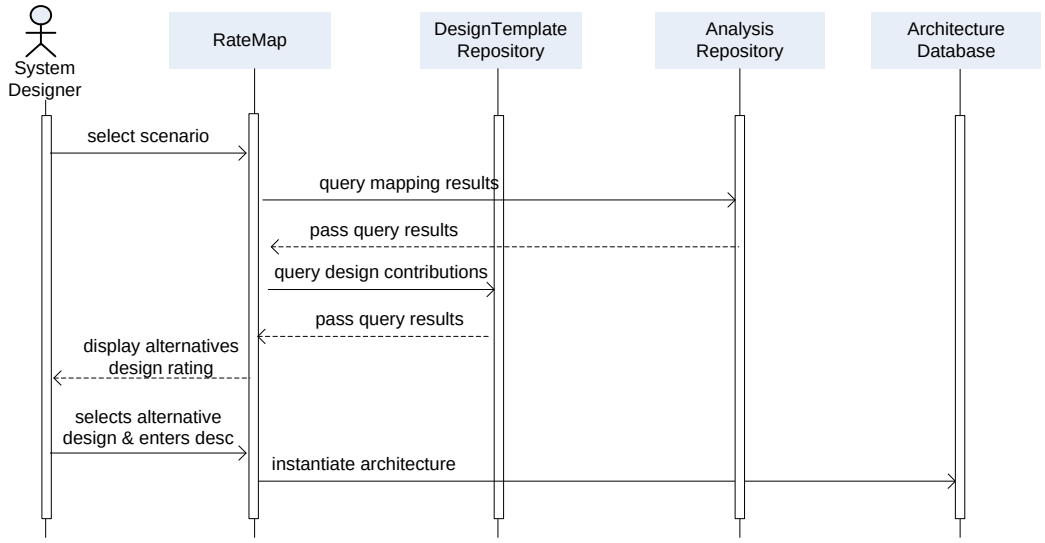


Fig. B1.5 Rate design sequence diagram

Table B1.6 Map services use-case description

CSAFE: Map Services	
Actor	System Designer, Analysis Repository
Description	1. System designer enters scenario name and service control requests results of selected design templates from architecture database. 2. Then, the system designer selects required alternative design and service control requests related design components from design template repository to be displayed in the list. 3. The system designer selects a service to map, again service control query design component details and submits the results back to the control. 4. Then, service control compare and match the selected service onto appropriate design component. 5. The results are store in architecture database and submit to service form to be displayed onto a panel by establishes a link between the service and the design component.
Data	Service (Non-functional requirement)
Stimulus	System designer selects service map template
Response	CSAFE stores component mapping results and a link is display onto a panel in the service form
Alternative flow of events	3.a. Matching design template not found. Indicate error message.

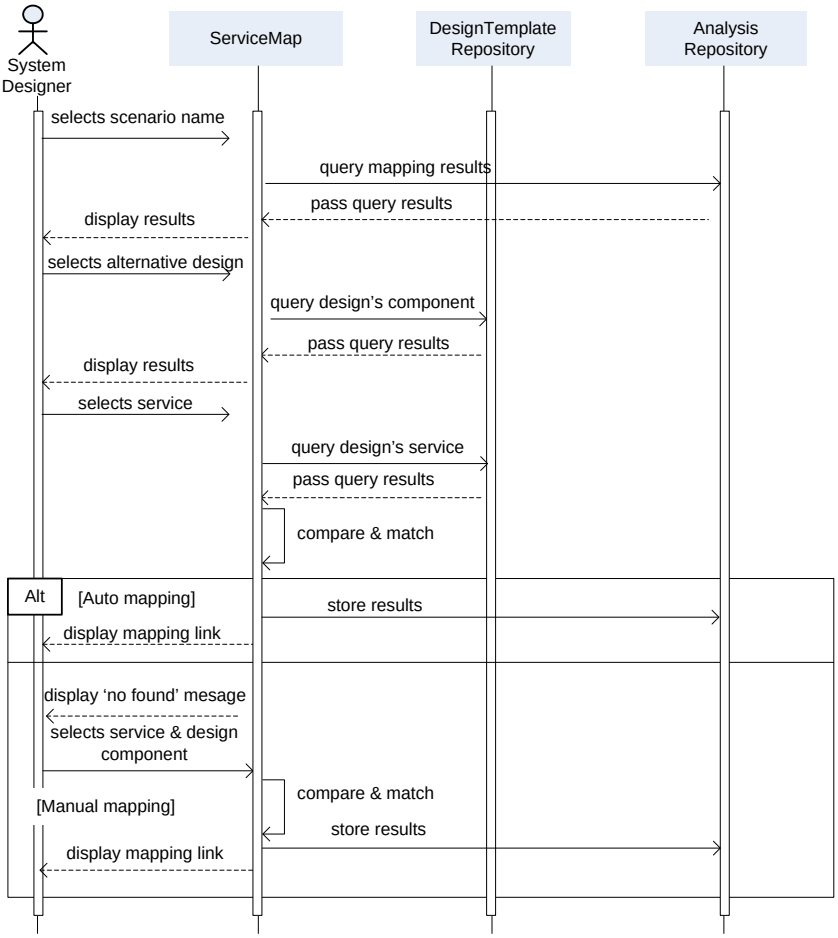


Fig. B1.6 Map services sequence diagram

Table B1.7 Map component use-case description

CSAFE: Map Components	
Actor	System Designer, Analysis Repository
Description	1. System designer enters scenario name and component control requests results of selected design templates from architecture database. 2. Then, the designer selects required alternative design and component control requests related design components from architecture database to be displayed in list on component form. 3. The system designer selects a component to map, component control query design component details and submits the results back to the control. 4. Then, service control compare and match the selected design component onto concrete component. 5. The results are store in architecture database and submit to component form to be displayed onto a panel by establishes a tag between the design component and the concrete component.
Data	Design component

Stimulus	System designer selects component map template
Response	CSAFE stores component mapping results and a tag is display in the component form
Alternative flow of events	4.a. Matching component not found. Indicate error message.

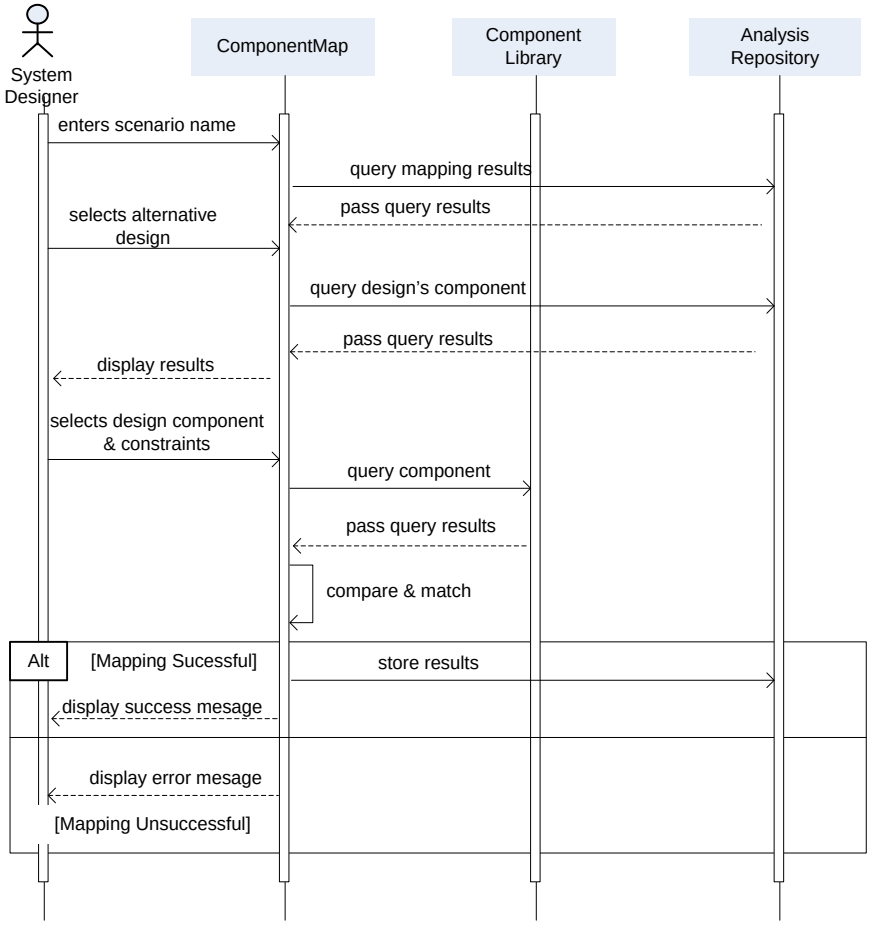


Fig. B1.7 Map components sequence diagram

Table B1.8 Assess architecture use-case description

CSAFE: Assess Architecture	
Actor	System Designer, Analysis Repository
Description	1. System designer enters scenario name and assess control requests formulated scenarios and its results from analysis repository. 2. Then, assess control submits a query for design contributions to template design library. 3. The query results are passes back to assess control. 4. Subsequently, mean values are calculated and the results are passes to assess template to be displayed.
Data	Scenario name

Stimulus	System designer selects assess template
Response	Analysis overall results are calculated and displayed.
Alternative flow of events	-

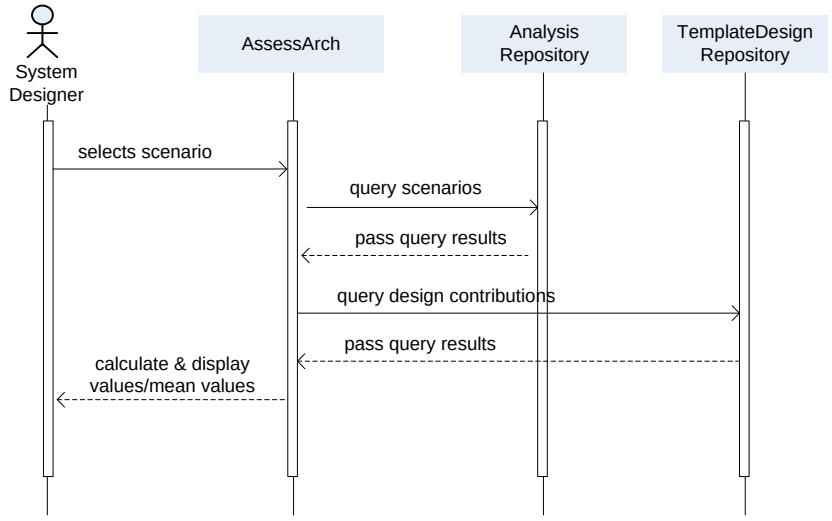


Fig. B1.8 Assess architecture sequence diagram

Table B1.9 Generate graphs use-case description

CSAFE: Generate Graphs	
Actor	System Designer, Analysis Repository
Description	1. System designer selects contribution level (e.g. level 1: best architectural designs, level 2: concern, level 3: sub-concern). 2. ContrGraph request data from Analysis Repository and calculate these dataset. 3. Contribution bar chart is display on assessment template. 4. System designer selects architectural design. 5. ScoreGraph request data from Analysis Repository and calculate these dataset. *6.* Scores pie charts are display on assessment template. *7.* System designer selects architectural design. 8. TradeOffGraph request data from Analysis Repository and calculate these dataset. 9. Component trade-off line chart is display on assessment template.
Data	Contribution graph dataset, Score graph dataset and Trade-off dataset.
Stimulus	System designer selects graph (i.e. contribution, scores, trade-off)
Response	Graph display on assessment template
Alternative flow of events	-

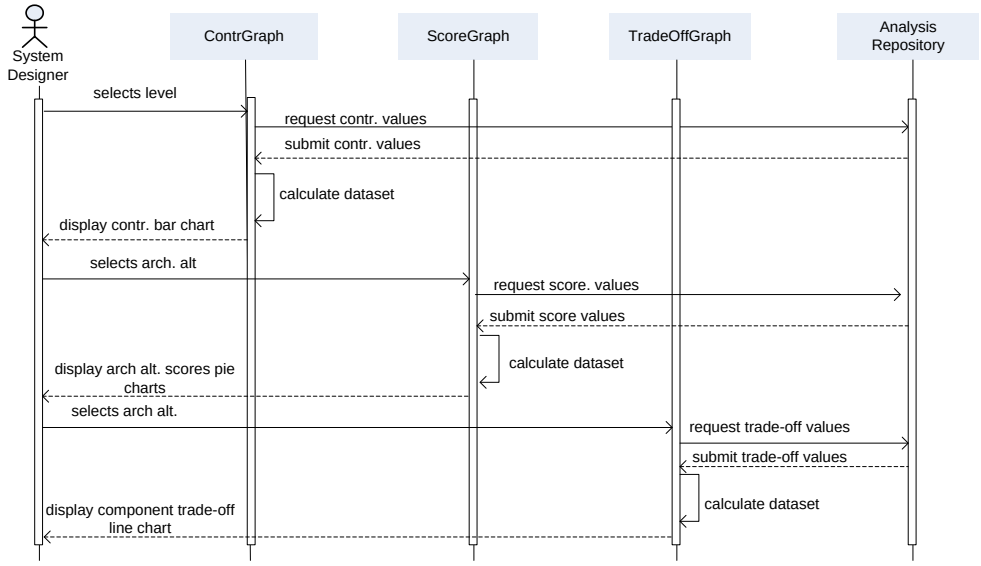


Fig. B1.9 Generate graphs sequence diagram

Table B1.10 Assess architecture use-case description

CSAFE: Generate report	
Actor	System Designer, Analysis Repository
Description	1. System designer selects a scenario. 2. Report request architectural design alternatives details from architecture database. 3. Report display report to the system designer. 4. System designer requests to print the report. 5. Report raster and print the report.
Data	Architectural design alternatives configurations
Stimulus	System designer selects report template
Response	Architectural design alternatives report is generated
Alternative flow of events	-

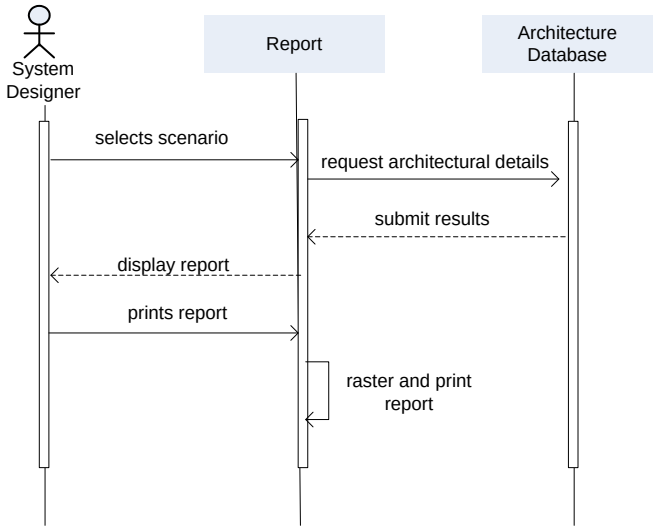


Fig. B1.10 Generate report sequence diagram

Table B1.11 Maintain rules repository use-case description

CSAFE: Maintain Rules Repository	
Actor	System Designer, XMI/XML Parser
Description	1. System designer browses and selects XMI/XML design template specification and clicks OK. 1. The XMI/XML parser parses the rule specification and checks against XMI/XML schema. 2. The XMI/XML parser stores rule descriptions in rules repository 3. The tool organizes the rule in tree hierarchy and each element detail description is display on the description form.
Data	XMI/XML design template specification
Stimulus	System designer selects design template manager template
Response	Design template library is updated
Alternative flow of events	1.a. The system designer selects a design template node. A confirmation message is display and upon confirmation the rule is removes from design template library.

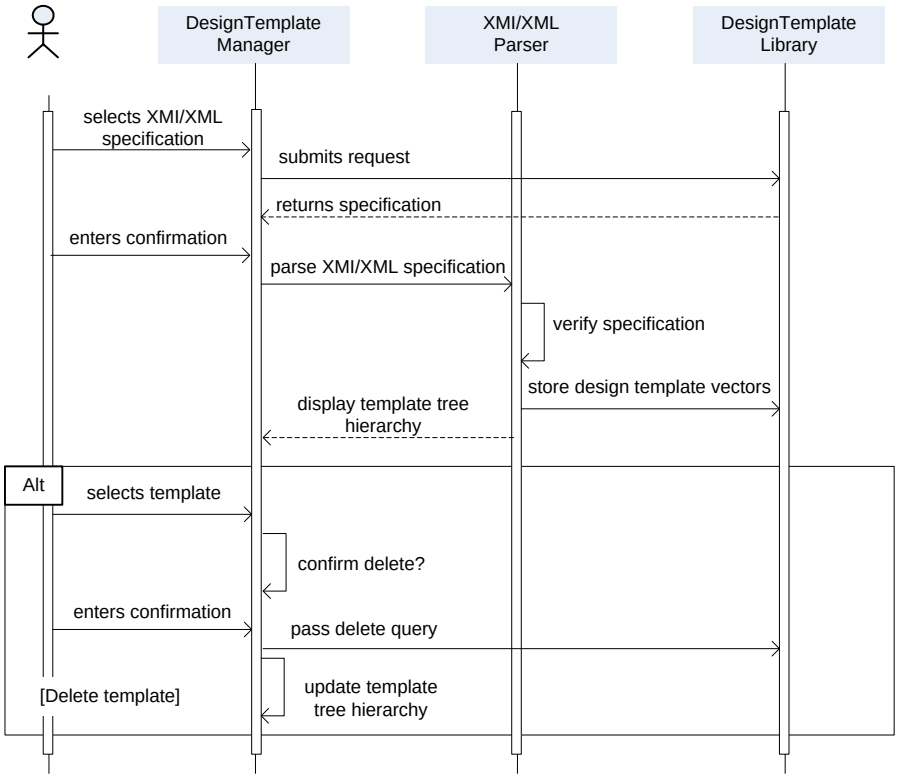


Fig. B1.11 Maintain Rules Repository sequence diagram

Table B1.12 Maintain Component Repository use-case description

CSAFE: Maintain Component Repository	
Actor	System Designer, XMI/XML Parser
Description	1. System designer browses and selects XMI/XML component specification and clicks OK. The parser parses the component specification and checks against XMI/XML schema. 2. The XMI/XML parser stores component descriptions in component repository 3. The tool organizes the component in tree hierarchy and each element detail description is display on the description form.
Data	XMI/XML component specification
Stimulus	System designer selects component manager template
Response	Component library is updated
Alternative flow of events	1.a. The system designer selects a component node. A confirmation message is display and upon confirmation the component is removes from component library.

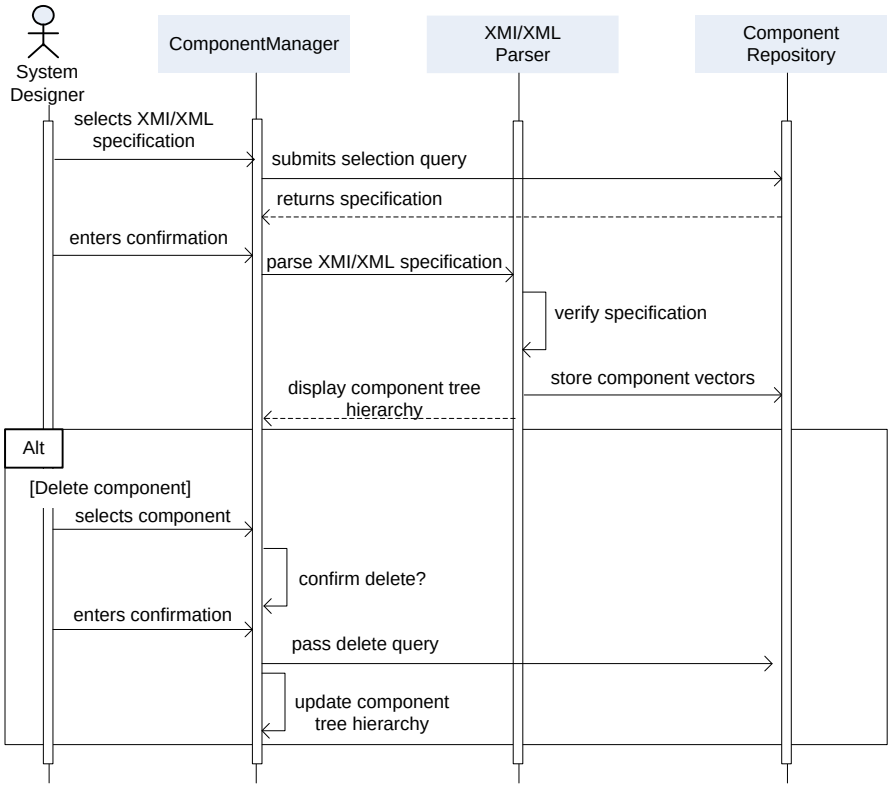


Fig. B1.12 Maintain Component Repository sequence diagram

B2. CSAFE Class Diagrams

CSAFE class diagram are shown in Fig. B2.1 – B.2.2.

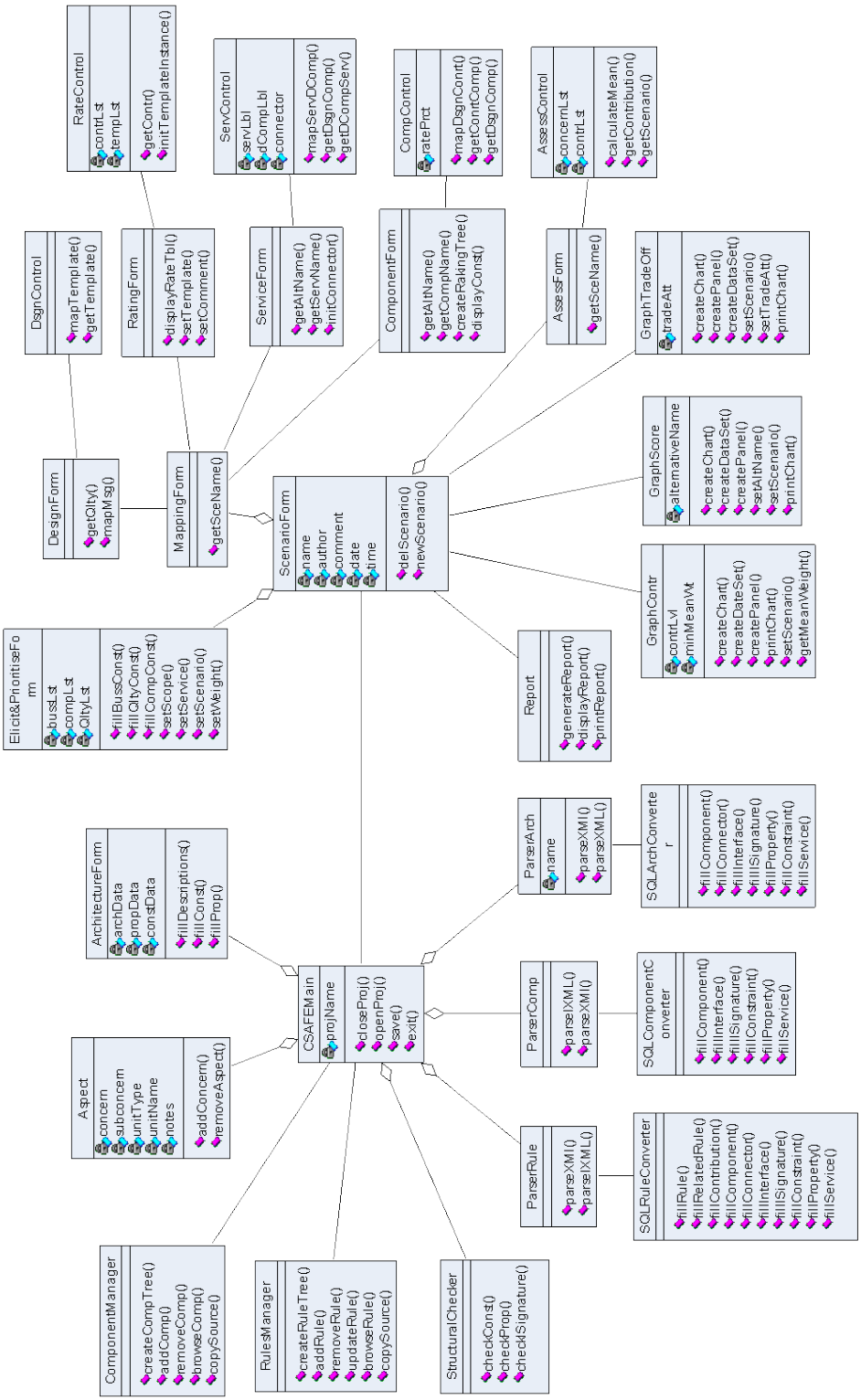


Fig. B2.1 CSAFE toolset project boundary and control class diagram

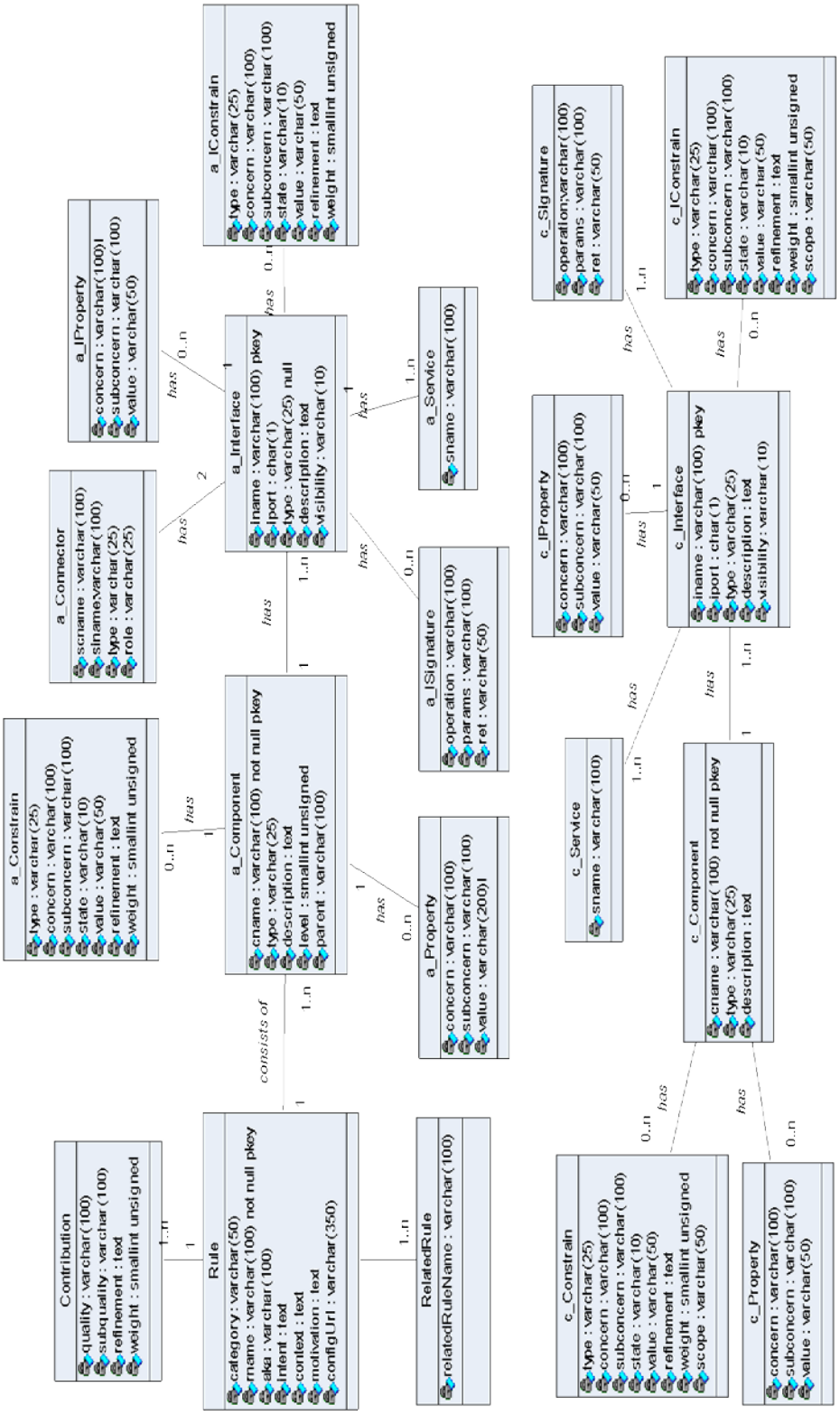


Fig. B2.2 CSAFE toolset project entity class diagram

Appendix C:

User Manual CSAFE Toolset

User Manual

for

CSAFE Toolset

Version 1.0

Prepared by Novia Admodisastro

Lancaster University

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

1.1 Purpose

This document describes detailed user manual for **C**omponent-based **S**oftware **A**rchitectural analysis **F**ram**E**work (CSAFE) Toolset. The CSAFE toolset has been developed to be as intuitive and easy to use as possible. Most functions in the system are obvious however this user manual aims to give a guide to performing the most common functions in the system.

1.2 Intended Audience and Reading Suggestions

This is a guidelines document meant for CSAFE users that may involve system architect, project manager, domain expert and programmer.

The following are the system and system features covered by this User Manual:

1. System requirements (hardware and software)
2. System features

2. System Requirements

CSAFE Toolset is written entirely in Java and therefore is platform independent. To successfully start and run CSAFE Toolset the following hardware and software require:

1. Java 5 is strongly recommended for Windows, Linux and Mac OS X platforms.
2. A specific operating system is not required. However, it has been predominantly developed and tested on Windows.
3. Database – MySQL Server 5.1 is used to access and to process data in the database.
4. Processor – 500MHz or higher processor (or compatible processor).
5. Memory – 512 MB RAM or higher
6. Hard Disk – 2 GB or higher

3. System Features

3.1 Main Windows

CSAFE main window consists of three parts: menu toolbar, a project area and a workspace area as shown in Fig. 1. The Project area contains a tree view of the system architecture which includes components, connectors, interfaces and so on.

The workspace area is tabbed with the specification and scenario panes, where the specification pane is use to display the architecture elements description and the scenario formulation pane is use to view and access formulated scenario.

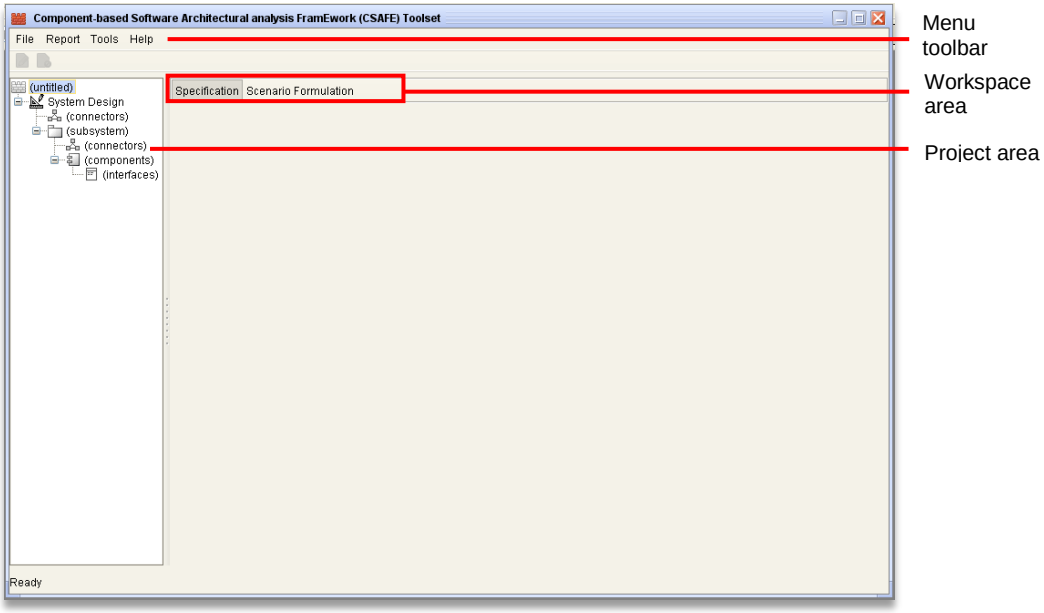


Fig. 1. CSAFE main window

3.2 Toolbar Menus

The menu toolbar provides quick access to used project configuration, architectural analysis managements and tool tutorials through File Menu, Report Menu, Tools Menu and Help Menu (refers to Fig. 2).

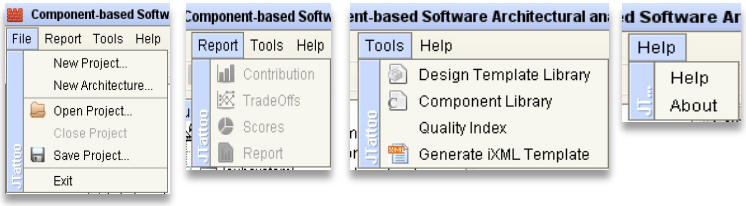
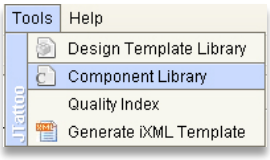


Fig. 2. File menu, Report menu, Tools menu and Help menu

3.3 Managing Component Library

Concrete software components specifications could be view and updates in the component library by clicking:



That brings up the Component Library window (refers to Fig. 3) which display a list of components on the left side and the component details of the right side.

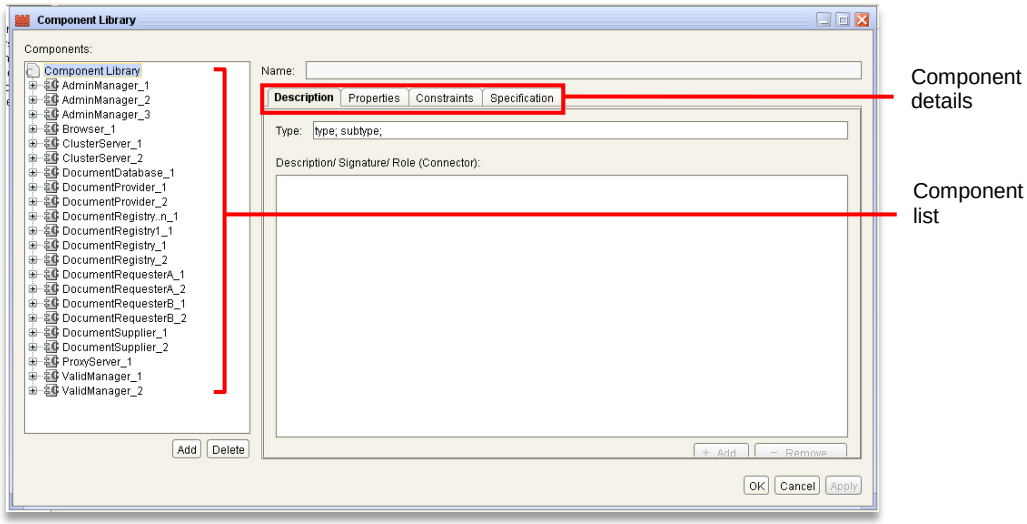


Fig. 3. Component Library window

A new component could be added by clicking:

This would display the 'New Component' dialog that requests for the component XML specification and name as shown in Fig. 4.

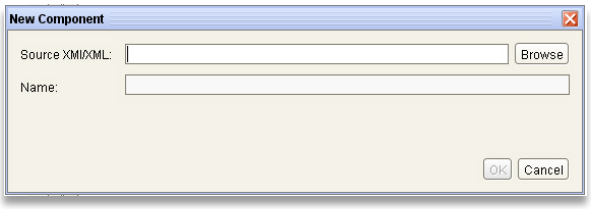


Fig. 4. 'New Component' dialog

Clicking the 'Browse' button assessing the component specification file where a file filter is implements to keep unwanted files from appearing in the directory listing (refers to Fig. 5.)

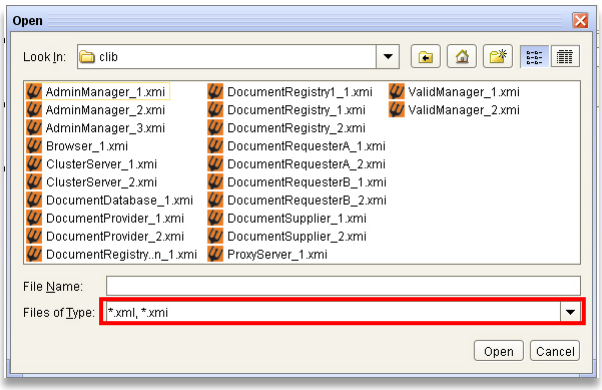


Fig. 5. Filter files for component specification

Error message are flagged by the XMI/XML parser when mismatched occurs such as duplicated component name, component specification is non-conformance to XML schema and etc. (refers to Fig. 6 and Fig. 7).

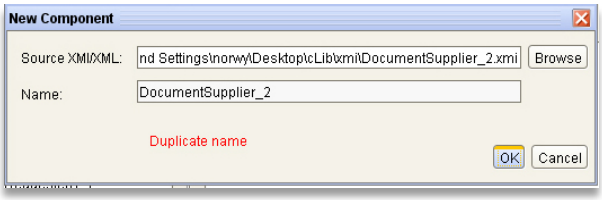


Fig. 6. Error message of component with duplicated name

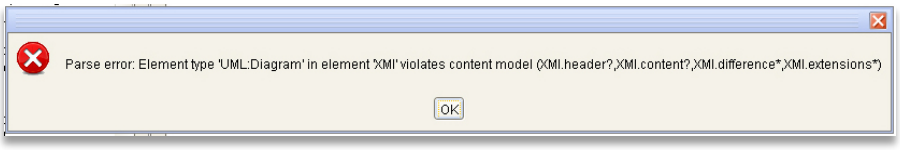


Fig. 7. Error message of non-conformance XML schema

A component could be deleted by selecting the component node on the tree and clicking:

and a delete dialog is display to confirm deletion as in Fig 8. After selecting 'Yes', the component is removed from the component tree.

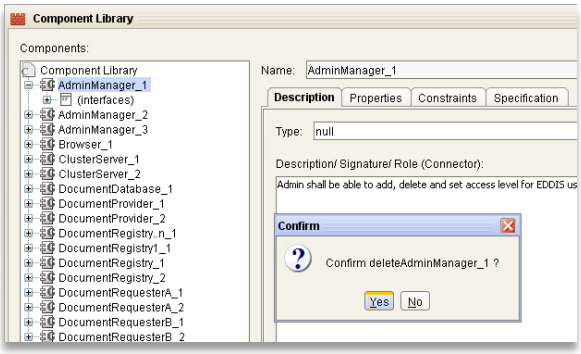


Fig. 8. Delete confirmation dialog

Component details such as its descriptions, properties, constraints and specification can be viewed by selecting a component node, for example the *AdminManager_1* details is shown in Fig. 9 – Fig.12.

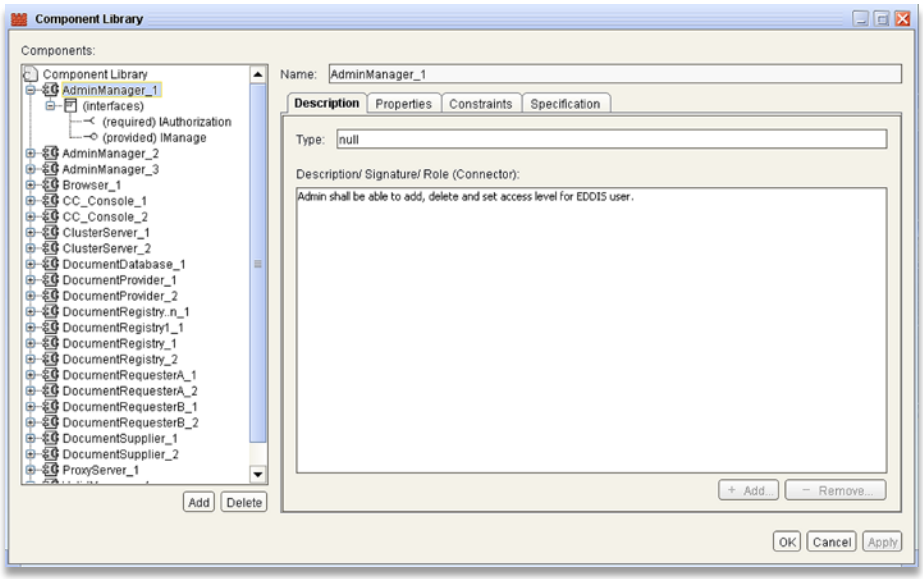


Fig. 9. AdminManager_1 descriptions

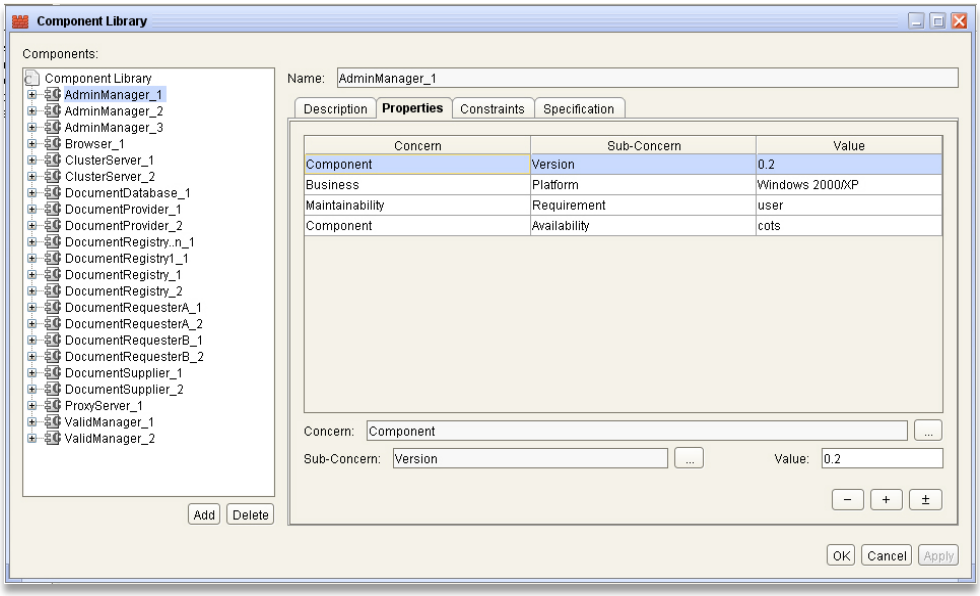


Fig. 10. *AdminManager_1* properties

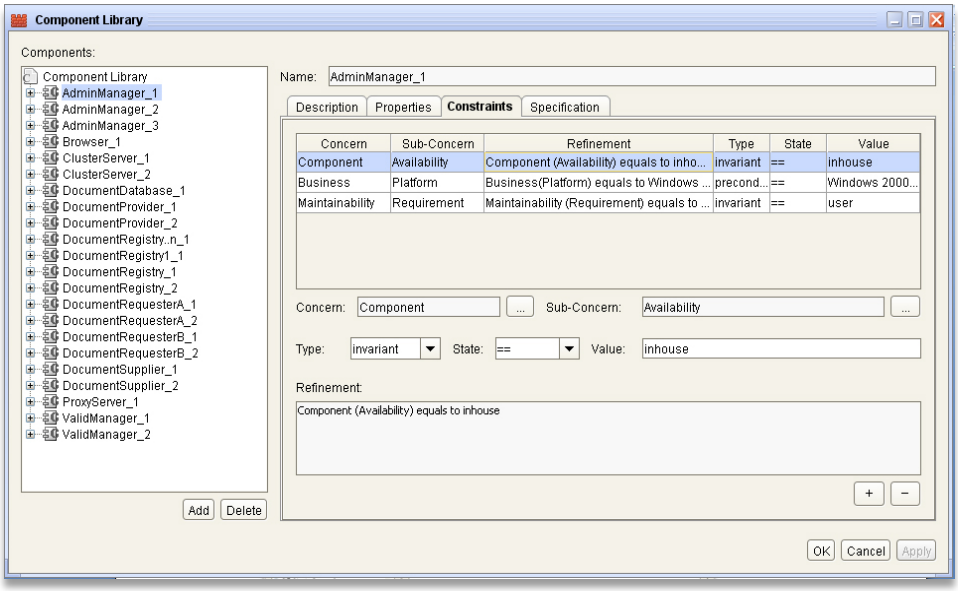


Fig. 11. *AdminManager_1* constraints

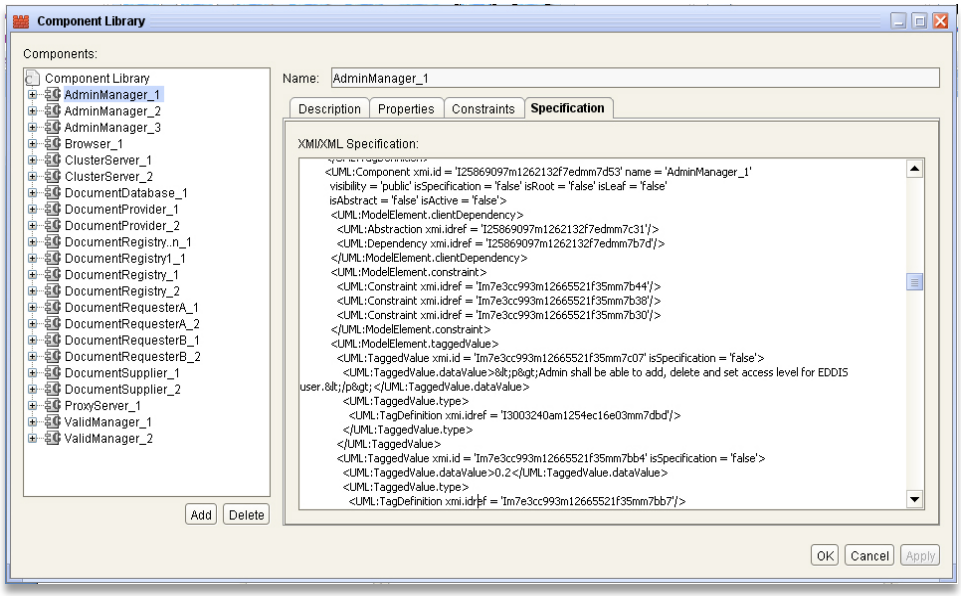
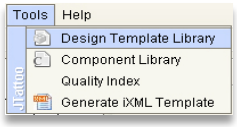


Fig. 12. *AdminManager_1* specification

3.4 Managing Design Template Library

Design template which consists of patterns, styles and local-schemes could be view and updates in the design template library by clicking:



That brings up the Design Template Library window (refers to Fig. 13) which display a list of design template which organised by its categories on the left side and the design template details of the right side.

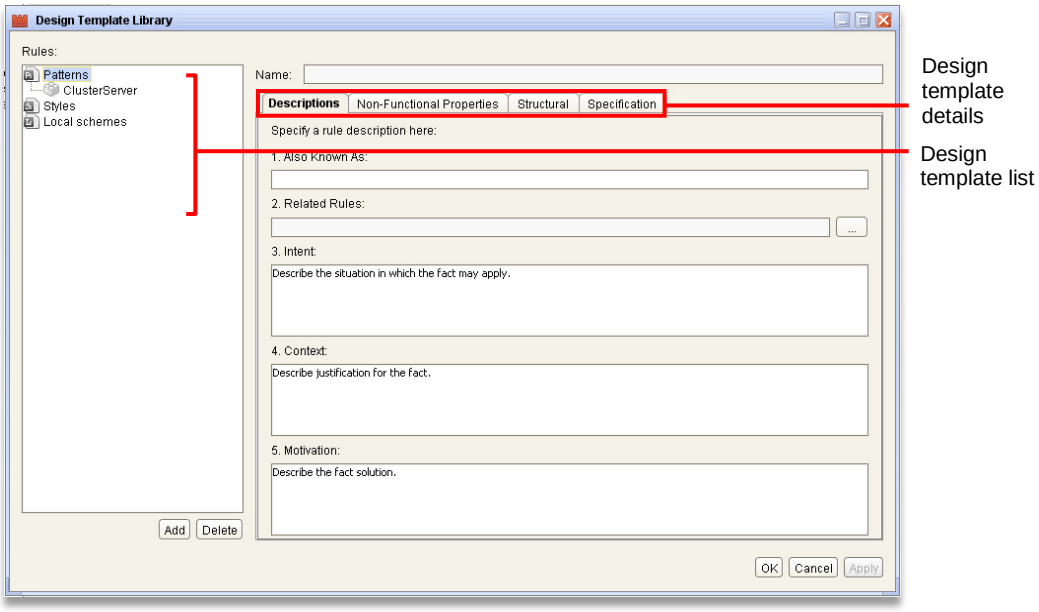


Fig. 13. Design template library window

A new design template could be added by clicking:

This would display the 'New Design Template' dialog that requests for the design template XML specification, name and category as shown in Fig. 14.

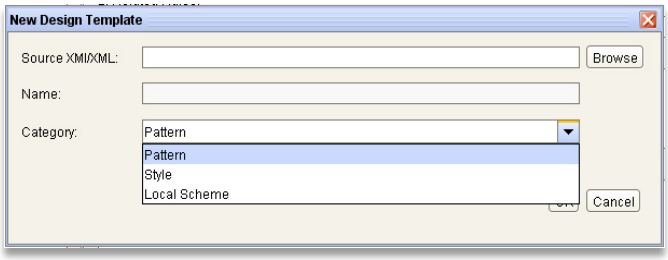


Fig. 14. 'New Design Template' dialog

Clicking the 'Browse' button assessing the design template specification file where a file filter is implements to keep unwanted files from appearing in the directory listing (refers to Fig. 15.)

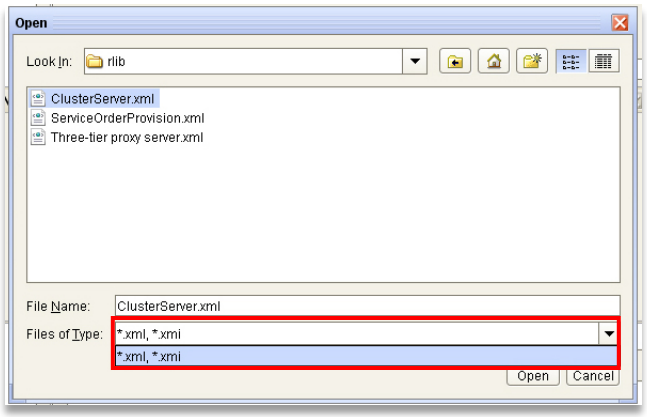


Fig. 15. Filter files for design template specification

Clicking 'OK' button in 'New Design Template' dialog added the design template to the template tree (refers to Fig. 16) considering that parsing is implemented successfully (no error).

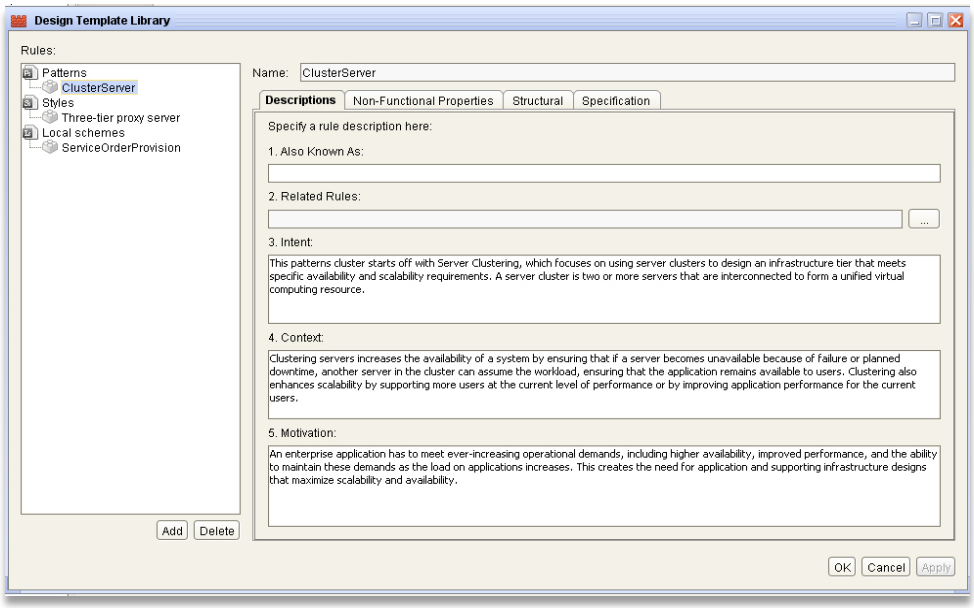


Fig. 16. *ClusterServer* pattern is organise in the design template tree

Error message are flagged by the XMI/XML parser when mismatched occurs such as for example duplicated design template name, design template specification is non-conformance to XML schema and etc. (refers to Fig. 17 – Fig. 20).

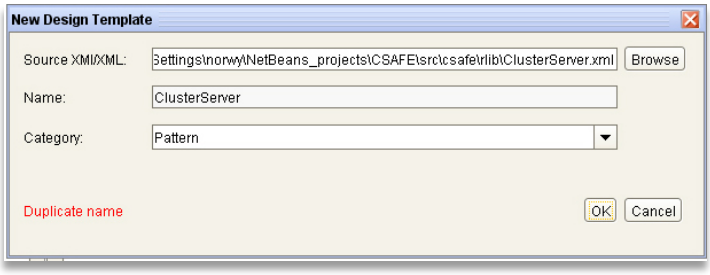


Fig. 17. Error message of design template with duplicated name

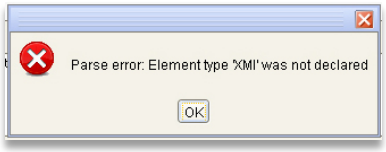


Fig. 18. Error message of non-conformance XML schema

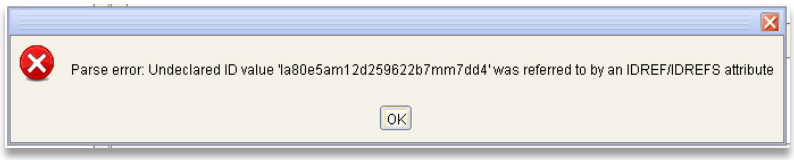


Fig. 19. Error message when missing template element in the specification

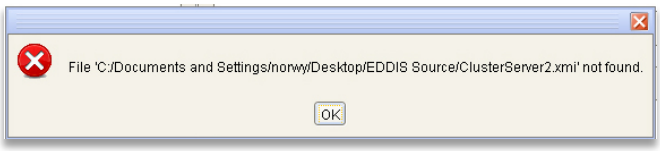


Fig. 20. Error message when XMI/XML schema was not found

A design template could be deleted by selecting the design template node on the tree and clicking:

and a delete dialog is display to confirm deletion as in Fig 21. After selecting ‘Yes’, the design template is removed from the template tree.

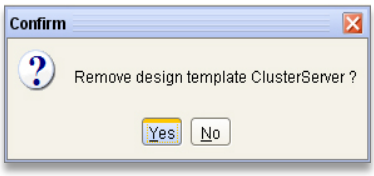


Fig. 21. Delete confirmation dialog

Design template details such its descriptions, contributions (non-functional properties), configuration and specification can be viewed by selecting a template node. For example Fig. 22 shows an example of *ClusterServer* design template descriptions.

If the design template is described using iXML description, its descriptions and contributions are automatically captured from the specification. However, if it is being described using XMI then these details are fills in manually. Fig. 22 shows *ClusterServer* pattern which being described using XMI, its descriptions are enter manually and follow by clicking 'Apply' button.

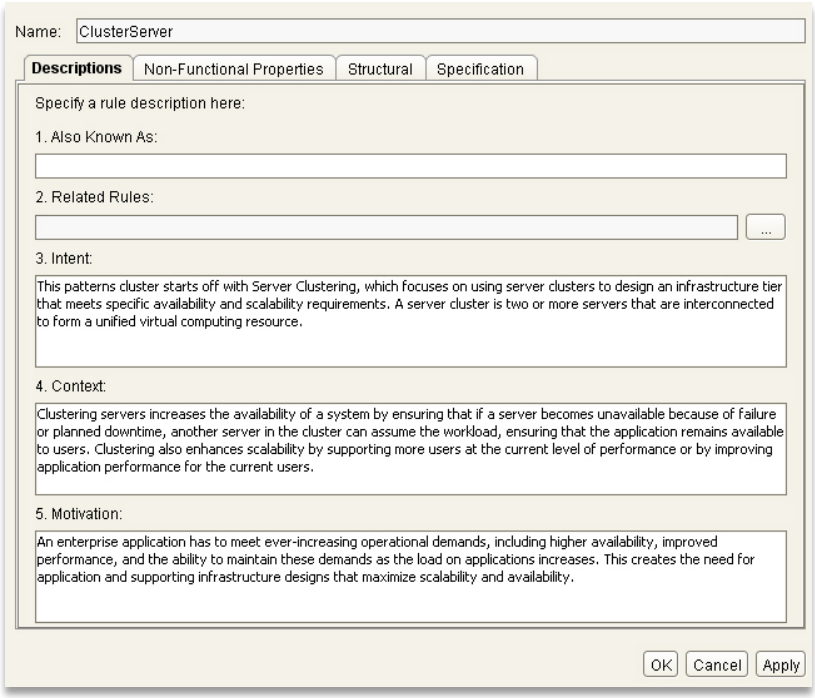


Fig. 22. *ClusterServer* pattern descriptions

Related Rules field is disabled, a value in enters by clicking:

This will open the ‘Design Template’ dialog, where we can choose reference to other closely related design templates as shown in Fig. 23.

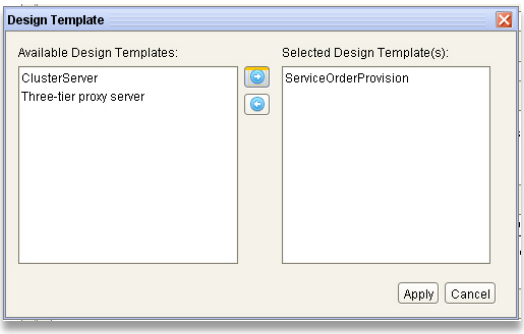


Fig. 23. ‘Design Template’ dialog

The second tab the dependency and contribution that template may possess shown in weighting factor. An example given is *ClusterServer* pattern as shown in Fig. 24.

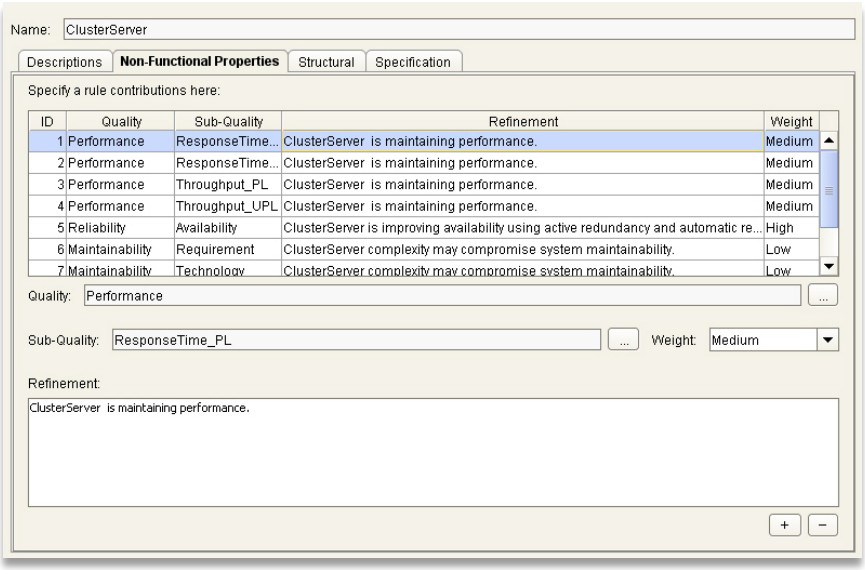


Fig. 24. *ClusterServer* pattern contributions

Again, since *ClusterServer* pattern is described in XMI its contributions are manually entered. Concern and sub-concern fields are disabled, values for these fields are enters by clicking:

This will open the ‘Quality Descriptions’ dialog, where we can choose related concern and sub-concern for the template and clicks ‘OK’ as shown in Fig. 25. Weighting factor of this contribution is selected from a weight drop down box as in Fig. 26.

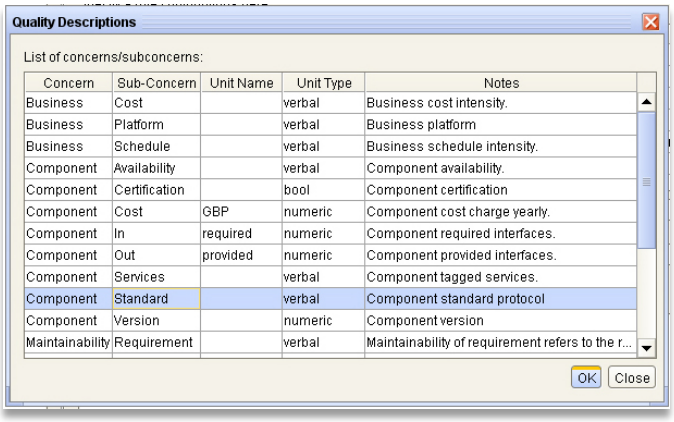


Fig. 25. Contribution of design template

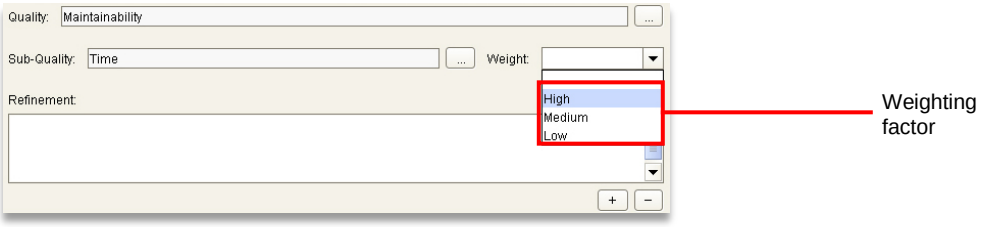


Fig. 26. Assigning weighting value for the contribution

The third tab the structural which illustrates template configuration. An example is *ClusterServer* pattern as shown in Fig. 27.

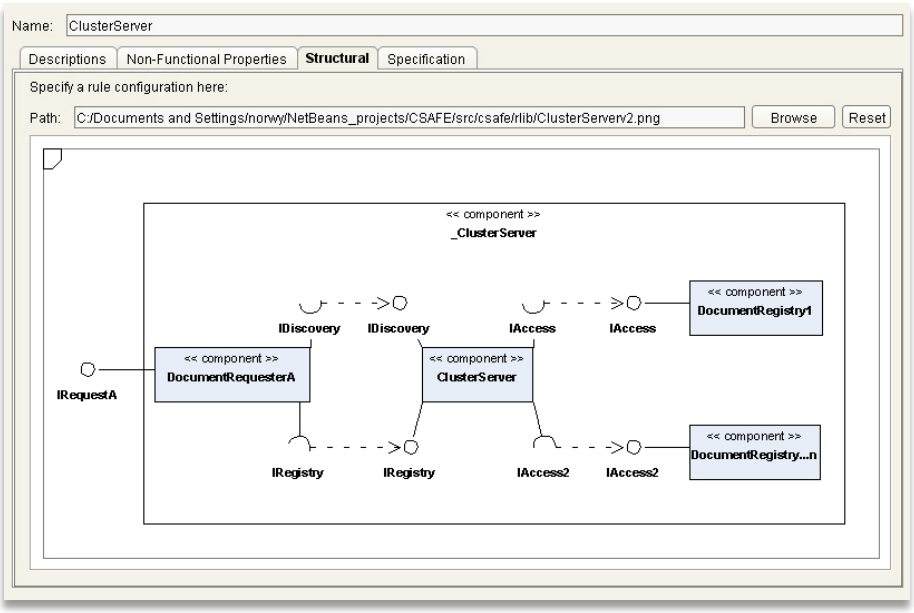


Fig. 27. *ClusterServer* pattern structural

Clicking

display 'Open' dialog which allows structural file to be retrieved as in Fig. 28.

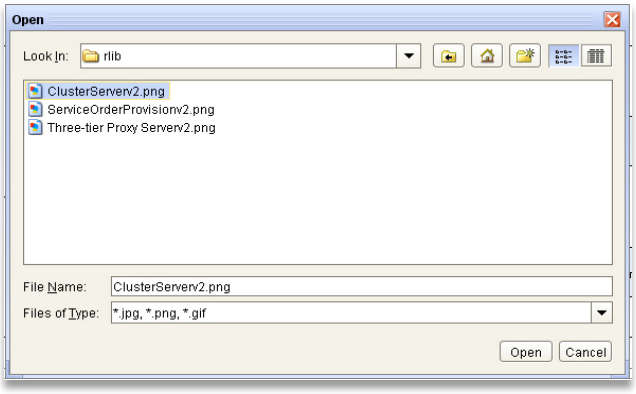


Fig. 28. Open dialog browse template structure file

The forth tab shows the design template specification. An example is *ClusterServer* pattern as shown in Fig. 29.

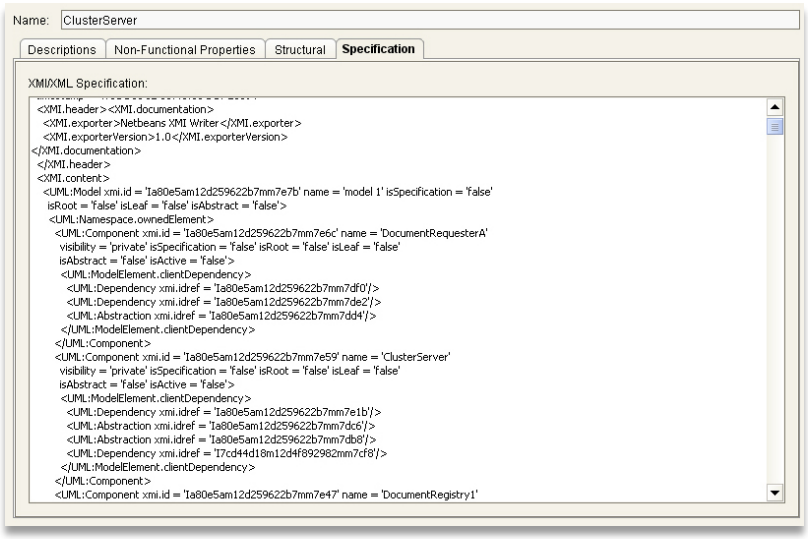
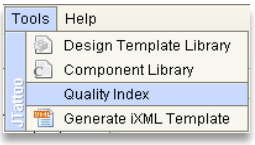


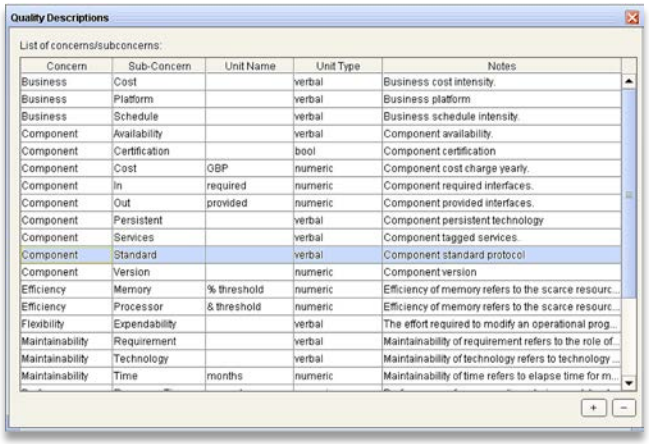
Fig. 29. *ClusterServer* pattern specification

3.5 Updating Quality Index List

List of quality could be view and updates in the Quality Index by clicking:



This brings up Quality Index window as in Fig. 30 that listed a quality descriptions such as its concern, sub-concern, unit name, unit type and notes.



Concern	Sub-Concern	Unit Name	Unit Type	Notes
Business	Cost		verbal	Business cost intensity.
Business	Platform		verbal	Business platform
Business	Schedule		verbal	Business schedule intensity.
Component	Availability		verbal	Component availability.
Component	Certification		bool	Component certification
Component	Cost	GBP	numeric	Component cost charge yearly.
Component	In	required	numeric	Component required interfaces.
Component	Out	provided	numeric	Component provided interfaces.
Component	Persistent		verbal	Component persistent technology
Component	Services		verbal	Component tagged services.
Component	Standard		verbal	Component standard protocol
Component	Version		numeric	Component version
Efficiency	Memory	% threshold	numeric	Efficiency of memory refers to the scarce resourc...
Efficiency	Processor	& threshold	numeric	Efficiency of memory refers to the scarce resourc...
Flexibility	Expendability		verbal	The effort required to modify an operational prog...
Maintainability	Requirement		verbal	Maintainability of requirement refers to the role of...
Maintainability	Technology		verbal	Maintainability of technology refers to technology ...
Maintainability	Time	months	numeric	Maintainability of time refers to elapse time for m...

Fig. 30. Quality index window

A new quality detail could be added into the list by clicking:

This will instantly provide a pop-up 'New Quality' dialog which requests for few details to be entered such as concern, sub-concern, unit name – optional (e.g. months, GBP etc.), unit type and notes – optional (refers to Fig. 31). We can type the first few letters of the concern and sub-concern, and the autocomplete will finish the entry (refers in Fig. 32).

Fig. 31. Quality unit type numeric, verbal or boolean

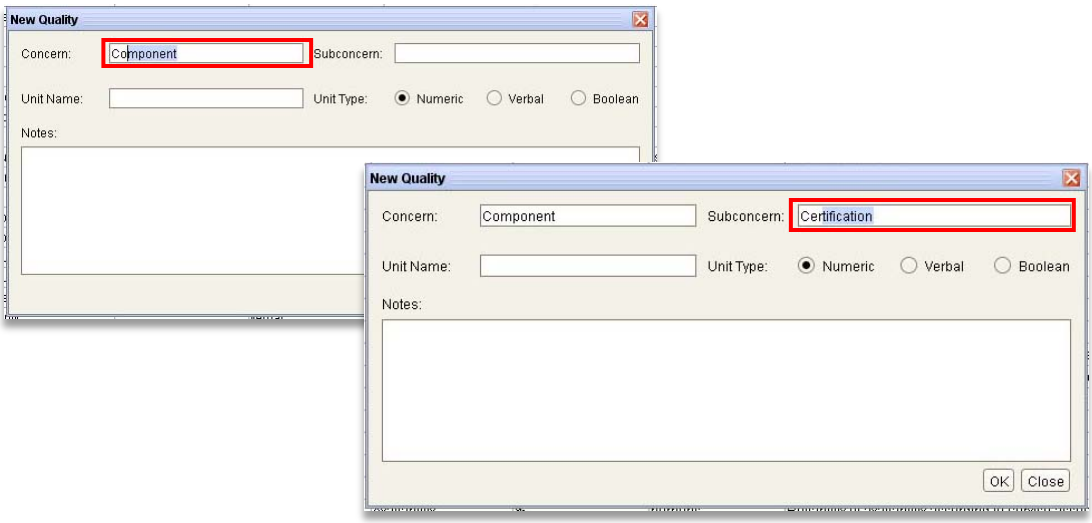


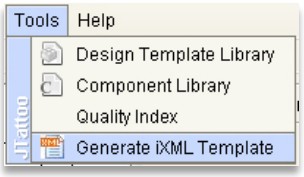
Fig. 32. Autocomplete feature for concern and sub-concern fields

Deleting a quality from the list is achieved in the same way by clicking:

A delete confonfirmation dialog is display, and upon accepting 'Yes' the quality will be removed from the list.

3.6 Generating iXML Template

iXML metamodel for architecture design and design template could be explored by clicking:



That brings up iXML template window as in Fig. 33 that described the metamodels including explanation of its elements. Template metamodel is displayed when design template is selected from the drop down list (Fig. 34).

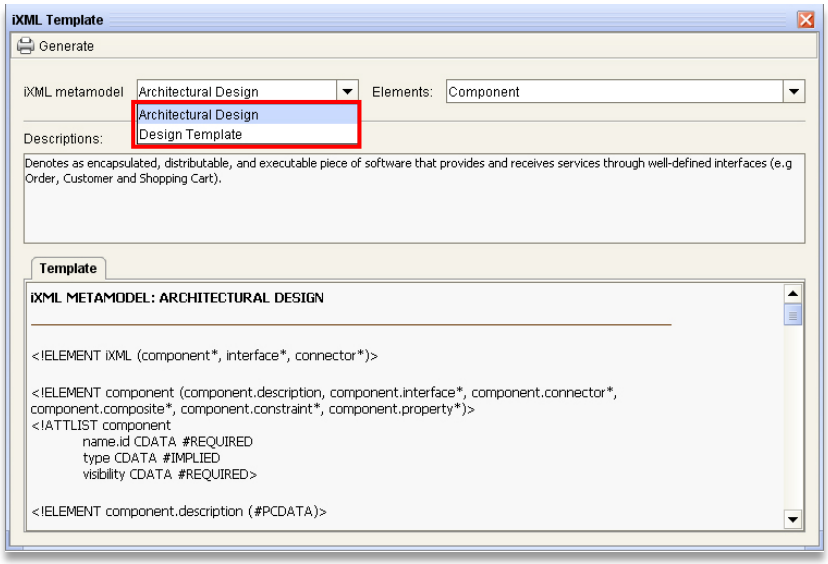


Fig. 33. iXML template window

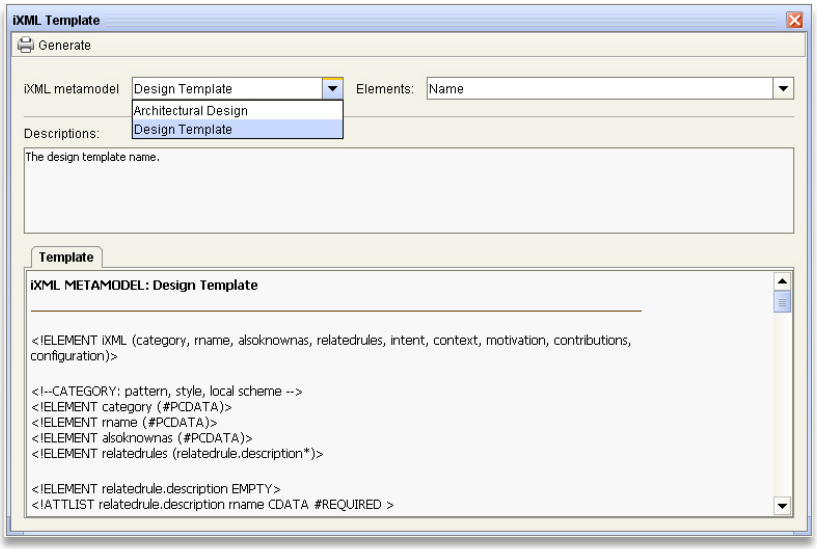


Fig. 34. iXML metamodel of design template

Architecture design and design template elements are shown respectively as in Fig. 35 and Fig. 36.

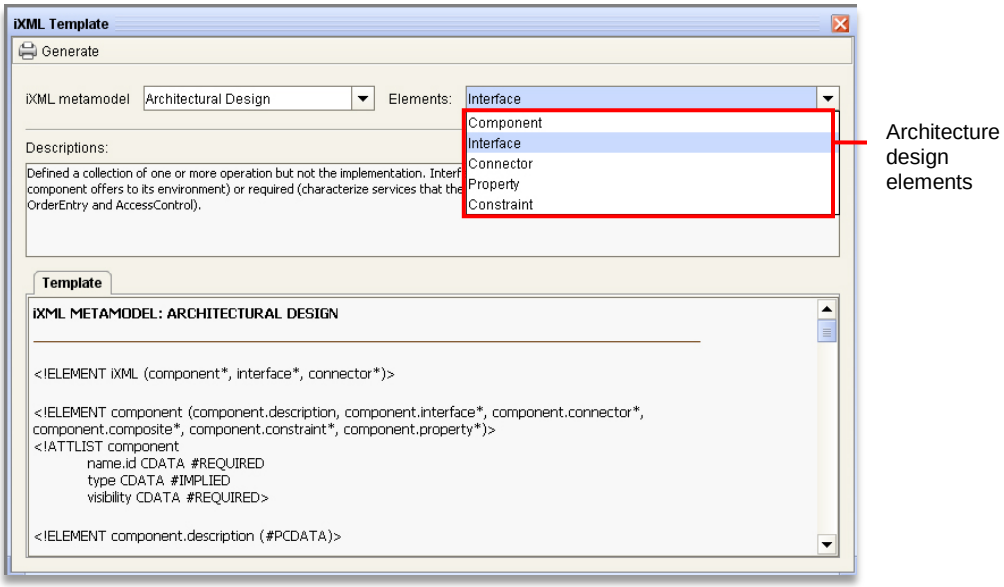


Fig. 35. Architecture design elements

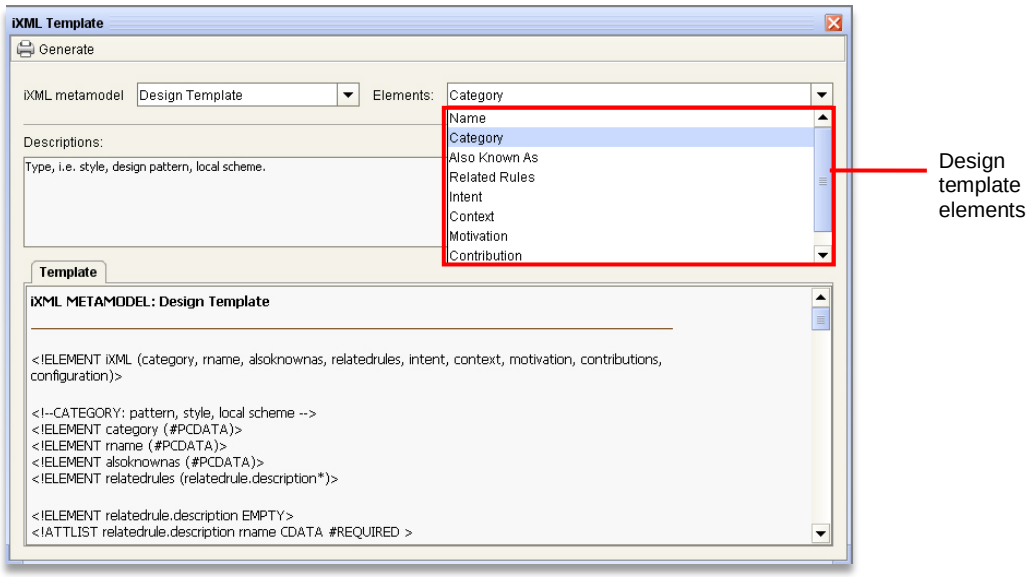


Fig. 36. Design template elements

Clicking

would allows the metamodel description to be saved (refers to Fig. 37).

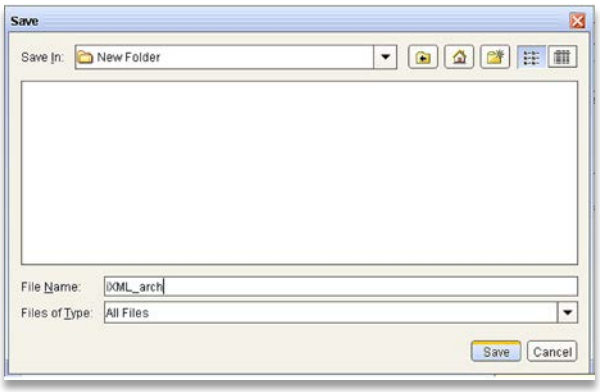


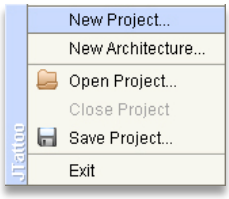
Fig. 37. Generates iXML template

3.7 Starting an Architectural Analysis Project

CSAFE provides two ways of conducting architectural analysis. The first takes as input an existing architecture and improves it through a process of structural,

quality and conformance analysis. The second assumes no architecture exists and uses the approach as a way of identifying possible starting architectural templates

Let starts with the first way, where an architecture design exists and going to be retrieved by clicking:



This opens a 'New Project' dialog which requires location of architecture design specification and project name as in Fig. 38.

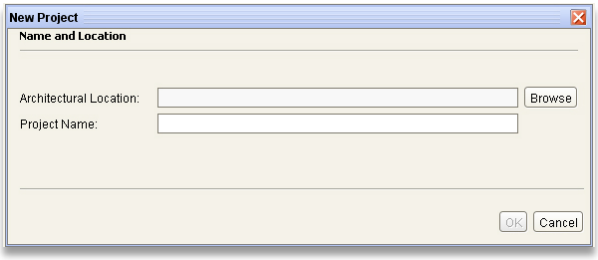


Fig. 38. 'New Project' dialog

Fig. 39 is an extension of locating process when Browse button it clicked. The Open dialog returns architecture specification path from a computer directory. File filter again is implemented, the filter only display XMI or XML architecture design specification format.

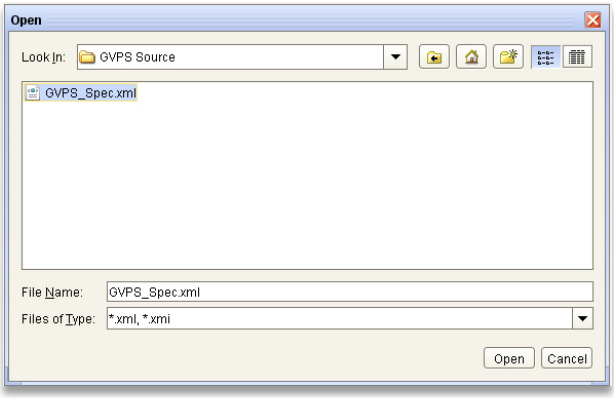


Fig. 39. Browse architecture design specification

If 'New Project' goes OK, meaning the parser successfully parsed the specification, a tree view of the architecture is displayed in the project area as in Fig. 40

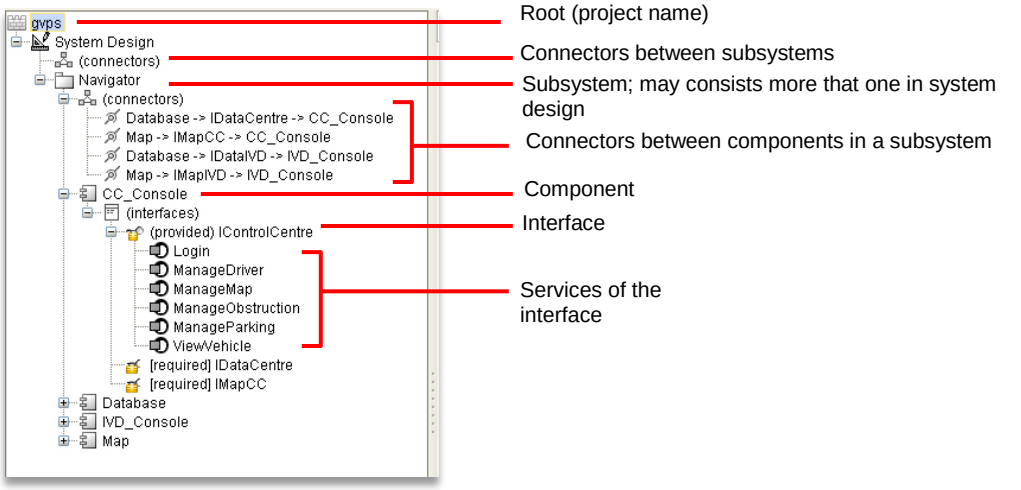


Fig. 40. Tree view of architecture design

The list of symbols employed by the interface is described below:

However, when mismatched occurs an error message is flagged by the parser. Examples of errors include missing referenced elements (e.g. service, component etc.), mismatched connector configuration, reference schema not found, specification is non-conformance to XML schema and etc. (refers to Fig. 41 – Fig. 43).

Fig. 41. Error message of non-conformance XML schema – tag found not being defined

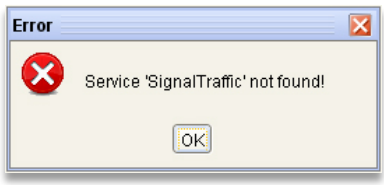


Fig. 42. Error message of missing architecture element – referenced service not in specification

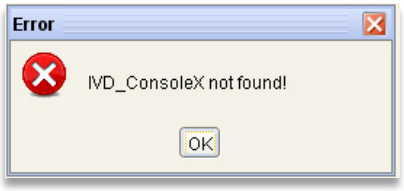
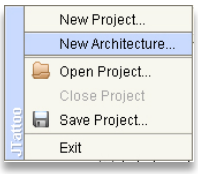


Fig. 43. Error message of mismatch configuration – referenced connector between components not valid

The second way for conducting an analysis is where we assume no architecture exists, clicks:



This opens the 'New Architecture Wizard' window (refers to Fig. 44). The wizard consists of two steps; firstly searching for suitable design templates as according to system goals, and secondly choose an appropriate design template and generates an architecture specification.

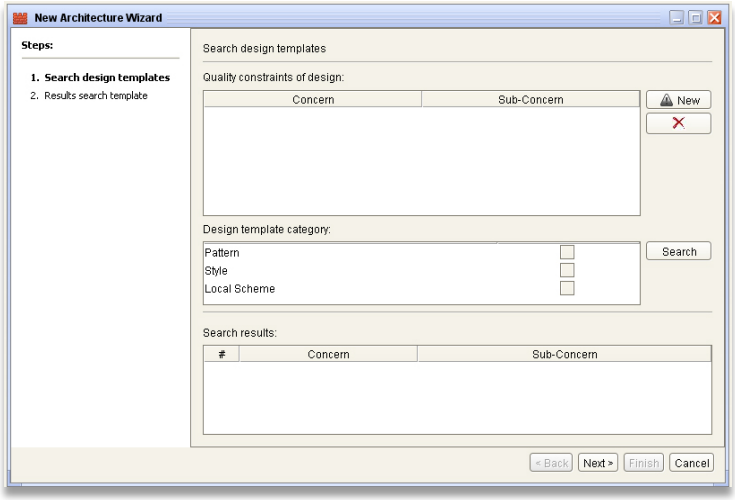


Fig. 44. New Architecture Wizard window

Clicks 

to enters system goals in to the quality design list. This would display 'Quality Descriptions' dialog as in Fig. 45.

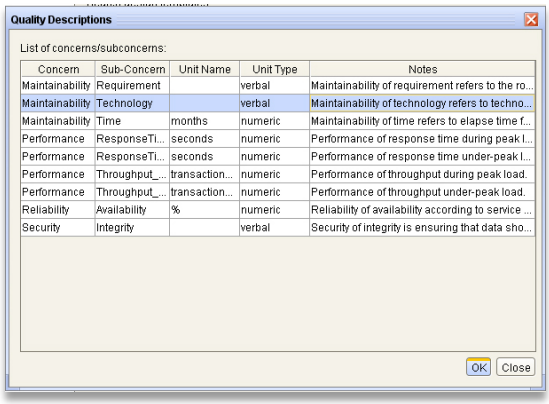


Fig. 45. Quality description dialog

Removing goal from the list is achieved by clicking;

When ‘Search’ button is clicks, searching is conducted. Status message is flagged (Fig. 46) and the results are displayed on lower panel (Fig. 47). Proceed ‘Next’ to Step 2.

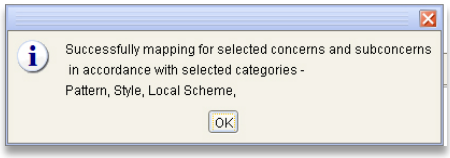


Fig. 46. Search result status message

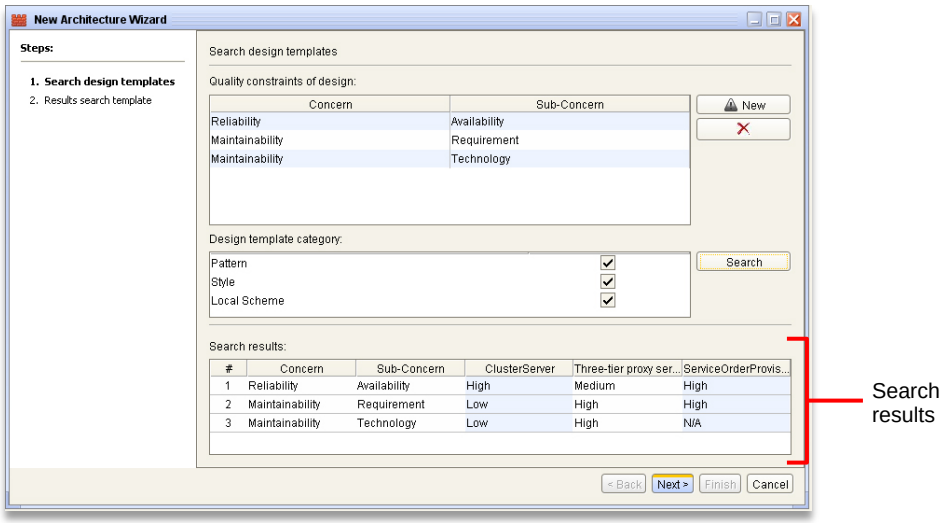


Fig. 47. Wizard step 1

Step 2 of the wizard requires project name to be entered and an architecture alternative to be selected (Fig. 48). Upon clicking 'Finish' the toolset generates specification for the selected design template under that project name. The project architecture design could be retrieved from Open dialog as in Fig. 49.

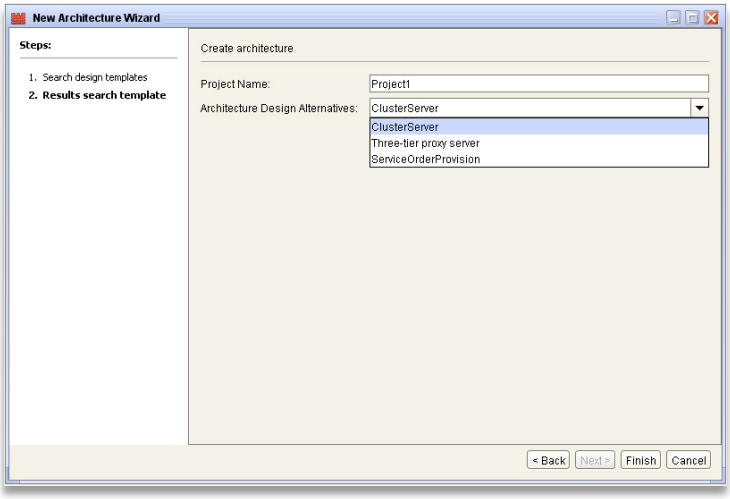


Fig. 48. Wizard step 2

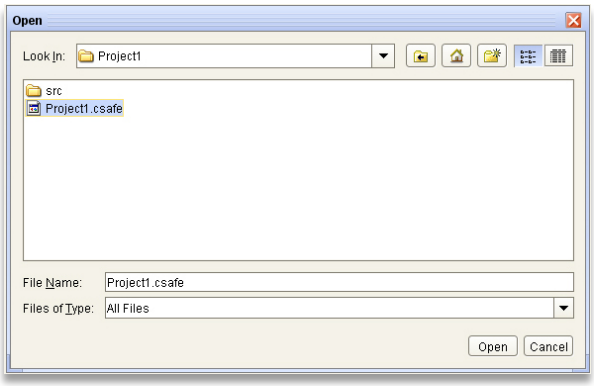


Fig. 49. Open dialog retrived generated speicication

3.8 About and Helps

CSAFE toolset provides assistant thru Help menu. Clicks:

brings Help window that firstly explained about CSAFE toolset (refers to Fig. 50). Simply proceed assiatnt by clicking next navigation button on the top panel to view contents of CSAFE tutorials Fig. 50. Fig. 51 is an example of step by step instruction for creating a new project.

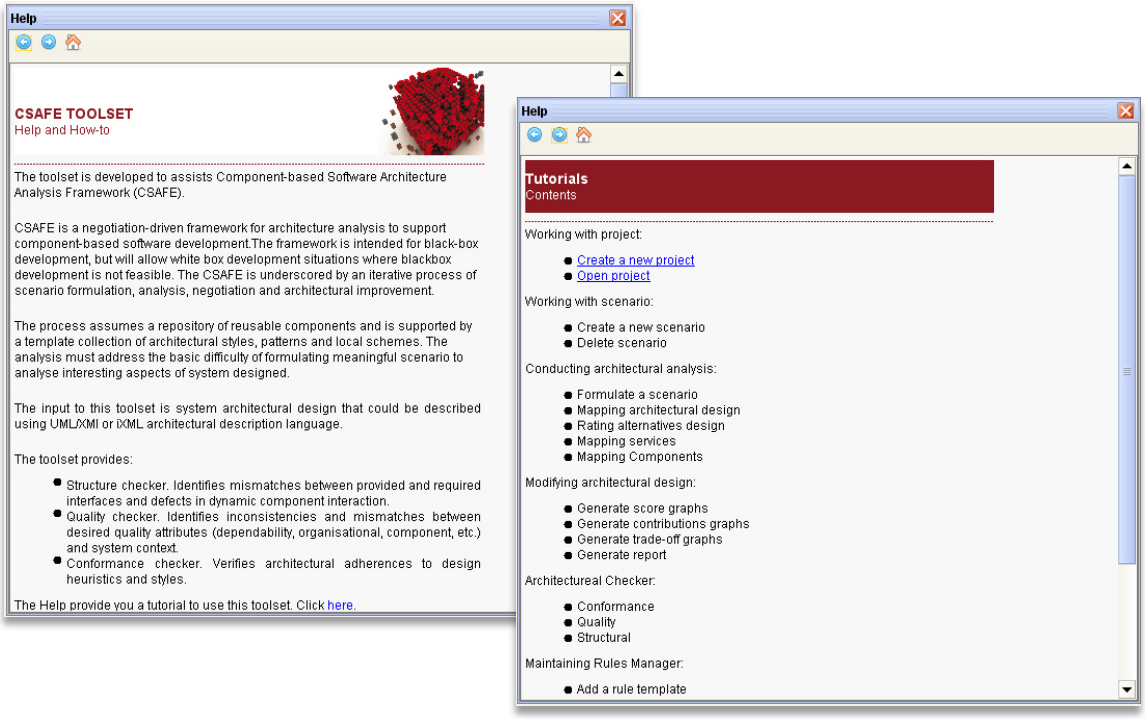


Fig. 50. Help window and CSAFE tutorial contents

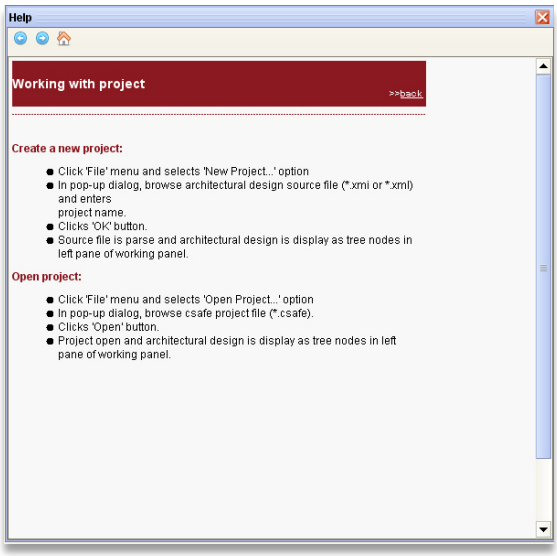


Fig. 51. Step by step tutorial

A quick information about CSAFE i.e version and contacts could be viewed by clicking:

This opens the About screen as in Fig. 52.

Fig. 52. About screen

Appendix D:

EDDIS Detail Specifications & Results

D1. Detail Requirements

EDDIS details requirements are described in Table D1.1.

Table D1.1 EDDIS details requirements

Viewpoint		Requirement			
ID	**Role/Type**	**ID**	**Description**	**Rationale**	**Ranking**
Vp_1	EDDIS_User (Operator)	R1.1	EDDIS users shall be able to login on to the system via a Web-based interface using valid usernames and passwords.	To provide a universal access to EDDIS services	Essential
		R1.2	Once logged in, EDDIS users will have access to a set of services determined by the permissions associated with their accounts.	To provide a simple mechanism for managing user account	Important
		R1.3	EDDIS shall allow users to search and identify documents, which interest them. A document search will initiated by a search criterion and a list of databases to be searched. The output will be a set of document identifiers.	Basic EDDIS functionality	Essential

Viewpoint		Requirement			
ID	**Role/Type**	**ID**	**Description**	**Rationale**	**Ranking**
		R1.4	EDDIS shall allow users to determine the location of documents. A documents locate service will be initiated by a set of document identifiers and the output shall be a set of location identifiers.	Basic EDDIS functionality	Essential
		R1.5	EDDIS user shall allow users to order documents. A document order will be initiated by a set of document and location identifiers. The output will be a set of order identifiers and electronic/hardcopy documents.	Basic EDDIS functionality	Important
Vp_2	EDDIS_ Administrator (Operator)	R2.1	EDDIS shall provide facilities for setting up and managing user accounts.	To provide a central EDDIS administration management.	Important
		R2.2	EDDIS shall allow admin to create account for EDDIS user. Creating a new account require user name, matrix/staff no. and user level e.g. Undergraduate, Postgraduate and Staff.	Basic EDDIS administration functionality	Essential
		R2.3	EDDIS shall allow admin to delete EDDIS user account. An account delete require matrix or staff no.	Basic EDDIS administration functionality	Important
		R2.4	EDDIS shall allow admin to assign access level for EDDIS user.	Basic EDDIS administration functionality	Essential
Vp_3	Document_ Registry (Component)	R3.1	EDDIS shall be able to access a centralized document registry to obtain document and location identifiers using the Z39.50 document retrieval standard.	Document retrieval standard used in document registry	Important
Vp_4	Document_ Supplier (Component)	R4.1	The document order client will be use the Z39.50 document retrieval standard.	Document retrieval standard used by document suppliers	Important
Vp_5	EDDIS_ Consortium (Organisation)	R5.1	The system shall run on Microsoft Windows 2000 and Windows XP.	Most users are likely to use a Windows-based PC to access EDDIS services.	Essential
		R5.2	The system shall be develop according to schedule and cost estimated.	Under relatively strict delivery date and budget.	Important

Viewpoint		Requirement			
ID	**Role/Type**	**ID**	**Description**	**Rationale**	**Ranking**
		R5.3	The system shall ensure that a reasonable level of performance is maintained across the services at all times.	To ensure that a reasonable level of performance is given to users.	Important
		R5.4	The system shall ensure that availability of service is given to EDDIS users accordingly.	To ensure that a reliable service is given to users.	Essential
		R5.5	The system shall ensure that it is easy to maintain that allow for graceful replacements or extensions of components.	To ensure EDDIS services are easily maintainable according to requirements.	Useful

D2. Service Descriptions

EDDIS service descriptions are shown in Table D2.1 – D2.6.

Table D2.1 User_validation service description

EDDIS: S1.1.1 User_validation	
Actors	EDDIS_User, EDDIS_Administrator
Description	1. EDDIS request login 2. EDDIS operators enters a login 3. Verify login against a set of username-password pairs in the database 4. If login is valid: 4.1 System initialise operator account permission else 4.2 System prompts the operator to re-enter login with three attempts.
Entry conditions	1. Login ∈ set of valid username-password pairs
Exit conditions	1. System reset use account permission 2. Closes operator account
Constraints	1. Service shall be maintained in 12 months time or less. 2. Service shall be maintained using updated technology. 3. Service shall be provided using only certified components. 4. Service shall be provided by inhouse components. 5. Service shall be provided using version 4.0 or greater components.

Table D2.2 Document_services service description

EDDIS: S1.2.1 Document_services	
Actors	EDDIS_User
Uses	User_validation
Extends	Document_search, Document_locate, Document_order
Description	1. EDDIS user enters a username and password 2. If username and password are valid: 2.1 System initialise user account permissions 2.2 Display the services available to the user else 2.3 System prompts the user to re-enter username and password
Entry conditions	1. Valid username 2. Valid password
Exit conditions	1. System reset use account permission 2. Closes user account
Constraints	1. Service shall have response time under peak load equals to or less than 0.75 seconds. 2. Service shall have response time peak load equals to or less than 4 seconds. 3. Service shall have throughput peak load equals to or less than 150 per second 4. Service shall be maintained according to user requirement. 5. Service shall be maintained in 18 months time or less. 6. Service shall be provided by inhouse components. 7. Service shall conform to Z39.50 document retrieval standard. 8. Service is provided by component which has 5 provided interfaces or less.

Table D2.3 Admin_services service description

EDDIS: S2.1.1 Admin_services	
Actors	EDDIS_Administrator
Uses	User_validation
Description	1. EDDIS user enters a username and password 2. If username and password are valid: 2.1 Display administrator managing options. 2.2 If create, add EDDIS user info 2.3 If delete, delete EDDIS user info 2.4 If manage, set EDDIS user access level else 2.5 System prompts the user to re-enter username and password
Entry conditions	1. Valid username 2. Valid password
Exit conditions	1. Closes administrator account
Constraints	2. Service shall be maintained according to user requirement. 3. Service shall be provided by in-house components. 4. Service shall be provided using version 0.3 or greater components.

Table D2.4 Document_search service description

EDDIS: S1.3.1 Document_search	
Actors	EDDIS_User, Document_Registry
Description	1. EDDIS user enters search criterion and a set of document databases 2. If document is found a set of document identifiers is displayed else a "document not found" message is displayed 3. Search criterion is retained in user workspace for future searches
Entry conditions	1. Document_search $\in$available_services 2. Document databases $\subseteq$ set of user permissible databases.
Exit conditions	1. System access conditions are reset
Constraints	1. Service conforms to Z39.50 document retrieval standard. 2. Service shall have an availability of 60 % or more between 8:00hr – 22:00hr Monday to Friday.

Table D2.5 Document_locate service description

EDDIS: S1.4.1 Document_locate	
Actors	EDDIS_User, Document_Registry
Description	1. EDDIS user enters document identifier and a set of catalogues 2. If location is found a set of locations identifiers is displayed else a "location not found" message is displayed 3. Location identifiers is retained in user workspace for future locates
Entry conditions	1. Document_locate $\in$available_services 2. Document catalogues $\subseteq$ set of user permissible catalogues
Exit conditions	1. System access conditions are reset
Constraints	1. Service conforms to Z39.50 document retrieval standard. 2. Service shall have an availability of 60 % or more between 8:00hr – 22:00hr Monday to Friday.

Table D2.6 Document_order service description

EDDIS: S1.5.1 Document_order	
Actors	EDDIS_User, Document_Supplier
Description	1. EDDIS user enters document identifier and location identifiers 2. If order is successful a set of order identifiers and electronic documents is displayed else a "document is not available" is displayed 3. Order is retained in user workspace for future order references
Entry conditions	1. Document_order $\in$available_services 2. Set of selected suppliers $\subseteq$ set of user permissible suppliers
Exit conditions	1. System access conditions are reset
Constraints	1. Service conform to Z39.50 document retrieval standard 2. Document order must be accompanied by a signed copyright acceptance form. 3. Service shall be maintained in 18 months time or less. 4. Service shall have an availability of 45% between 8:00hr – 18:00hr weekday Monday to Friday.

D3. Constraint Descriptions

EDDIS constraint descriptions are shown in Table D3.1.

Table D3.1 EDDIS constraint descriptions

Concern	Sub-concern	ID	Description	Rationale	Scope
1-Performance	1-Response time under peak load	C.1.1.1	Under-peak load transaction complete is equals or less than 0.75 seconds	To ensure that a reasonable level of performance is given to users.	document _services
	2-Response time peak load	C.1.2.1	During peak load transaction complete is equals or less than 4 seconds.	To ensure that a reasonable level of performance is given to users.	document _services
	3-Throughput peak load	C.1.3.1	At peak load system is able to complete 150 transactions per second.	To ensure that a reasonable level of performance is given to users.	document _services
2-Reliability	1-Availability	C.2.1.1	System service has an availability of 60 % or more between 8:00hr – 22:00hr Monday to Friday.	To ensure sufficient access time.	document _search, document _locate; document _registry
		C.2.1.2	System service has an availability of 45% between 8:00hr – 18:00hr weekday Monday to Friday.	To ensure sufficient access time.	document _order; document _supplier
3-Maintainability (Trigger)	1-Time	C.3.1.1	Maintainable every 12 months or less.	Critical components	user_ validation
		C.3.1.2	Maintainable every 18 months or less.	Moderate components	document _order; document _supplier
		C.3.1.3	Maintainable every 18 months or less.	Moderate components	document _services
	2-Technology	C.3.2.1	Adapting to current technologies.	Security technology updated.	user_ validation
	3-Requirement	C.3.3.1	Configuration based on user requests.	To ensure services gracefully replace or extendable.	document _services

Concern	Sub-concern	ID	Description	Rationale	Scope
		C.3.3.2	Configuration based on user requests.	To ensure services gracefully replace or extendable.	admin_ services
4-Business	1-Cost	C.4.1.1	System development according to cost estimated	Under strict budget	System
	2-Schedule	C.4.2.1	System development according to schedule estimated.	Under strict delivery date	System
	3-Platform	C.4.3.1	The system shall run on Microsoft Windows 2000 and Windows XP	Most users are likely to use a Windows-based PC to access EDDIS services	System
5-Component	1-Availability	C.5.1.1	Document_supplier subscribe to available web services	Materials provided by 3rd party which not available in local library	document _registry
		C.5.1.2	Document_registry subscribe to available web services	Materials provided by 3rd party which not available in local library	document _supplier
		C.5.1.3	System services is in-house build	Components available in local library.	user_ validation,
		C.5.1.4	System services is in-house build	Components available in local library.	admin_ services
		C.5.1.5	System services is in-house build	Components available in local library.	document _services
	2-Standard	C.5.2.1	Service conforms to Z39.50 document retrieval standard.	Document retrieval standard used in document registry	document _registry
		C.5.2.2	Service conforms to Z39.50 document retrieval standard.	Document retrieval standard used in document supplier	document _supplier
		C.5.2.3	Service conform to Z39.50 document retrieval standard	Document retrieval standard used in document registry/supplier	document _services

Concern	Sub-concern	ID	Description	Rationale	Scope
	3-Cost	C.5.3.1	Subscription cost less than 500GBP yearly	Under strict budget	document _registry
		C.5.3.2	Subscription cost less than 650GBP yearly	Under strict budget	document _supplier
	4-Version	C.5.4.1	Component version is greater than or equals to 4.0.	Updated services	user_ validation
		C.5.4.2	Component version is greater than or equals to 0.3	Updated services	admin_ services
	5-In	C.5.5.1	The service requires five or less services.	To reduce EDDIS document services complexity	document _services
	6-Certification	C.5.6.1	Trusted services are required	To provide reliable user access.	user_ validation
		C.5.6.2	Trusted services are required	To provide reliable document discovery.	document _registry
		C.5.6.3	Trusted services are required	To provide reliable document provider.	document _supplier

D4. Concrete Component Descriptions

Concrete component descriptions described as the following:

AdminManager_1

Properties:	
Component(Version)	0.2
Component(Availability)	Cots
Maintainability(Requirement)	User
Interfaces:	
IManage: Admin_services, acct_create, acct_remove, acct_setaccess	
addUser	name:String, id:String, category:integer
deleteUser	id:String
setAccess	id:String

AdminManager_2

Properties:	
Component(Version)	0.4
Component(Availability)	Inhouse
Maintainability(Requirement)	User
Interfaces:	
IManage: Admin_services, acct_create, acct_remove, acct_setaccess	
addUser	name:String, id:integer, category:integer
deleteUser	id:integer
setAccess	id:integer

AdminManager_3

Properties:	
Component(Version)	0.4
Component(Availability)	Inhouse
Maintainability(Requirement)	User
Interfaces:	
IManage: Admin_services, acct_create, acct_remove, acct_setaccess	
addUser	name:String, id:integer, category:integer
deleteUser	id:integer
setAccess	id:integer

Browser_1

Properties:	
Component(Version)	2.0
Maintainability(Technology)	updated
Maintainability(Requirement)	user

ClusterServer_1

Properties:	
Component(Availability)	inhouse
Component(Certification)	no
Component(Version)	1.0
Component(Services)	accessSearch, accessLocate
Component(Standard)	Z39.50
Interfaces:	
IRegistry: accessLocate	
setLocate	docID:integer
IDiscovery: accessSearch	
setSearch	author:String, title:String

ClusterServer_2

Properties:	
Component(Availability)	inhouse
Component(Certification)	no
Component(Version)	2.0
Component(Services)	accessSearch, accessLocate
Component(Standard)	Z39.50
Interfaces:	
IRegistry: accessLocate	
setLocate	docID:integer
IDiscovery: accessSearch	
setSearch	author:String, title:String

DocumentDatabase_1

Properties:	
Component(Availability)	web service
Component(Certification)	no
Component(Version)	2.0
Component(Services)	accessSearch, accessLocate, accessOrder
Component(Standard)	Z39.50
Component(Cost)	400
Maintainability(Requirement)	vendor
Reliability(Availability)	45
Maintainability(Technology)	updated
Maintainability(Requirement)	vendor
Maintainability(Time)	12
Interfaces:	
IDatabase: accessSearch, accessLocate, accessOrder	
setSearch	author:String, title:String
setLocate	docID:String
setOrder	locID:String

DocumentRequesterA_1

Properties:	
Component(Version)	0.2
Component(Availability)	inhouse
Component(In)	4
Component(Out)	2
Component(Standard)	Z39.50
Maintainability(Time)	24
Maintainability(Requirement)	user
Reliability(Availability)	55
Performance(ResponseTime_UPL)	0.5
Performance(ResponseTime_PL)	4
Performance(Throughput_PL)	125
Interfaces:	
IRequestA: setSearch, setLocate	
Search	author:String, title:String
Locate	docID:String

DocumentRequesterA_2

Properties:	
Component(Version)	0.2
Component(Availability)	inhouse
Component(In)	4
Component(Out)	2
Component(Standard)	Z39.50
Maintainability(Time)	18
Maintainability(Requirement)	user
Reliability(Availability)	65
Performance(ResponseTime_UPL)	0.5
Performance(ResponseTime_PL)	3.5
Performance(Throughput_PL)	160
Interfaces:	
IRequestA: setSearch, setLocate	
Search	author:String, title:String
Locate	docID:String

DocumentRequesterB_1

Properties:	
Component(Version)	0.2
Component(Availability)	inhouse
Component(In)	4
Component(Out)	2
Component(Standard)	Z39.50
Maintainability(Time)	24
Maintainability(Requirement)	user
Reliability(Availability)	55
Performance(ResponseTime_UPL)	0.5
Performance(ResponseTime_PL)	4

Performance(Throughput_PL)	125
Interfaces:	
IRequestB: document_services, document_search, document_locate, document_order	
Search	author:String, title:String
Locate	docID:String
Order	locID:String

DocumentRequesterB_2

Properties:	
Component(Version)	0.2
Component(Availability)	inhouse
Component(In)	4
Component(Out)	2
Component(Standard)	Z39.50
Maintainability(Time)	18
Maintainability(Requirement)	user
Reliability(Availability)	65
Performance(ResponseTime_UPL)	0.5
Performance(ResponseTime_PL)	3.5
Performance(Throughput_PL)	160
Interfaces:	
IRequestB: document_services, document_search, document_locate, document_order	
Search	author:String, title:String
Locate	docID:String
Order	locID:String

DocumentSupplier_1

Properties:	
Component(Availability)	web service
Component(Certification)	yes
Component(Standard)	Z39.50
Component(Cost)	600
Component(Version)	4.1
Component(Services)	accessOrder
Reliability(Availability)	85
Maintainability(Technology)	updated
Maintainability(Requirement)	vendor
Interfaces:	
ISupplier: accessOrder	
setOrder	locID:String

DocumentSupplier_2

Properties:	
Component(Availability)	web service
Component(Certification)	yes
Component(Standard)	Z39.50
Component(Cost)	500

Component(Version)	4.1
Component(Services)	accessOrder
Reliability(Availability)	85
Maintainability(Technology)	updated
Maintainability(Requirement)	vendor
Interfaces:	
ISupplier: accessOrder	
setOrder	locID:integer

DocumentRegistry_1

Properties:	
Component(Availability)	web service
Component(Certification)	No
Component(Version)	2.0
Component(Services)	accessSearch,accessLocate
Component(Standard)	Z39.50
Component(Cost)	400
Maintainability(Requirement)	Vendor
Reliability(Availability)	75
Interfaces:	
ISearch: accessSearch	
setSearch	author:String, title:String
ILocate: accessLocate	
setLocate	docID:integer

DocumentRegistry_2

Properties:	
Component(Availability)	web service
Component(Certification)	No
Component(Version)	2.0
Component(Services)	accessSearch,accessLocate
Component(Standard)	Z39.50
Component(Cost)	700
Maintainability(Requirement)	Vendor
Reliability(Availability)	75
Interfaces:	
ISearch: accessSearch	
setSearch	author:String, title:String
ILocate: accessLocate	
setLocate	docID:integer

DocumentRegistry1_1

Properties:	
Component(Availability)	web service
Component(Certification)	No
Component(Version)	2.0
Component(Services)	accessSearch,accessLocate
Component(Standard)	Z39.50
Component(Cost)	400

Maintainability(Requirement)	Vendor
Reliability(Availability)	75
Interfaces:	
IAccess: accessSearch, accessLocate	
setSearch	author:String, title:String
setLocate	docID:integer

DocumentRegistry...n_1

Properties:	
Component(Availability)	web service
Component(Certification)	No
Component(Version)	2.0
Component(Services)	accessSearch,accessLocate
Component(Standard)	Z39.50
Component(Cost)	700
Maintainability(Requirement)	Vendor
Reliability(Availability)	75
Interfaces:	
IAccess: accessSearch, accessLocate	
setSearch	author:String, title:String
setLocate	docID:String

DocumentProvider_1

Properties:	
Component(Availability)	web service
Component(Certification)	yes
Component(Standard)	Z39.50
Component(Cost)	500
Component(Version)	4.1
Component(Services)	accessOrder
Reliability(Availability)	85
Maintainability(Technology)	updated
Maintainability(Requirement)	vendor
Interfaces:	
IOrder: accessOrder	
setOrder	locID:String

DocumentProvider_2

Properties:	
Component(Availability)	web service
Component(Certification)	No
Component(Standard)	Z39.80
Component(Cost)	750
Component(Version)	4.1
Component(Services)	accessOrder
Reliability(Availability)	85
Maintainability(Technology)	Updated

Maintainability(Requirement)	Vendor
Interfaces:	
IOrder: accessOrder	
setOrder	locID:String

ProxyServer_1

Properties:	
Component(Certification)	Yes
Maintainability(Technology)	Updated
Maintainability(Requirement)	User
Interfaces:	
IRequestC: confConn	
Access	

ValidManager_1

Properties:	
Component(Availability)	Inhouse
Component(Certification)	Yes
Component(In)	3
Component(Out)	3
Component(Version)	2.0
Component(Services)	user_validation,acct_permission
Business(Platform)	Windows 2000/XP
Maintainability(Time)	10
Maintainability(Technology)	updated
Interfaces:	
ILogin: User_validation	
validateUser	username:String, pwd:String
Logout	
IAuthorization: Acct_permission	
setLogin	isLogin:Boolean
resetCondition	

ValidManager_2

Properties:	
Component(Availability)	inhouse
Component(Certification)	yes
Component(In)	3
Component(Out)	3
Component(Version)	2.0
Component(Services)	user_validation,acct_permission
Maintainability(Time)	18
Maintainability(Technology)	updated
Interfaces:	
ILogin: User_validation	
validateUser	username:String, pwd:String
Logout	

IAuthorization: Acct_permission	
setLogin	isLogin:Boolean
resetCondition	

D5. SMART

Quality concern weights and design template contributions calculated using SMART in Table D5.1 for Scenario 1 of EDDIS.

Table D5.1 SMART for EDDIS - Scenario 1

Concern	Sub-Concern	Scope	S1: *ClusterServer (CS)*				S2: *Service-Order Provision (SOP)*				S3: *Three-tier proxy server (TPS)*			
			s.	s*		μ	s.	s*		μ	s.	s*		μ
Performance	ResponseTime_UPL	S1.2.1	2	0.08	0.20	0.641	2	0.08	0.21	0.818	2	0.08	0.21	0.795
	ResponseTime_PL	S1.2.1	2	0.08			2	0.08			2	0.08		
	Throughput_PL	S1.2.1	2	0.05			2	0.05			2	0.05		
Reliability	Availability	S1.3.1	3	0.12	0.30		3	0.12	0.31		2	0.08	0.21	
	Availability	S1.4.1	3	0.12			3	0.12			2	0.08		
	Availability	S1.5.1	3	0.08			3	0.08			2	0.05		
Maintainability	Requirement	S1.2.1	1	0.01	0.14		3	0.04	0.30		3	0.04	0.38	
	Requirement	S2.1.1	1	0.01			3	0.04			3	0.04		
	Technology	S1.1.1	1	0.03			0	0.00			3	0.08		
	Time	S1.1.1	1	0.03			3	0.08			3	0.08		
	Time	S1.5.1	1	0.04			3	0.12			3	0.12		
	Time	S1.2.1	1	0.01			3	0.04			3	0.04		

Quality concern weights and design template contributions calculated using SMART in Table D5.2 for Scenario 2 of EDDIS.

Table D5.2 SMART for EDDIS - Scenario 2

Concern	Sub-Concern	Scope	S1: *ClusterServer (CS)*				S2: *Service-Order Provision (SOP)*				S3: *Three-tier proxy server (TPS)*			
			s.	s*		μ	s.	s*		μ	s.	s*		μ
Performance	ResponseTime_UPL	S1.2.1	2	0.22	0.66	0.666	2	0.22	0.667	0.666	2	0.22	0.667	0.666
	ResponseTime_PL	S1.2.1	2	0.22			2	0.22			2	0.22		
	Throughput_PL	S1.2.1	2	0.22			2	0.22			2	0.22		

D6. Reports

Report snippets of the results and analysis process generated by the toolset are shown in Fig. D6.1. Full reports extracted from the report pane for Scenario 1 is shown in Table D6.1.

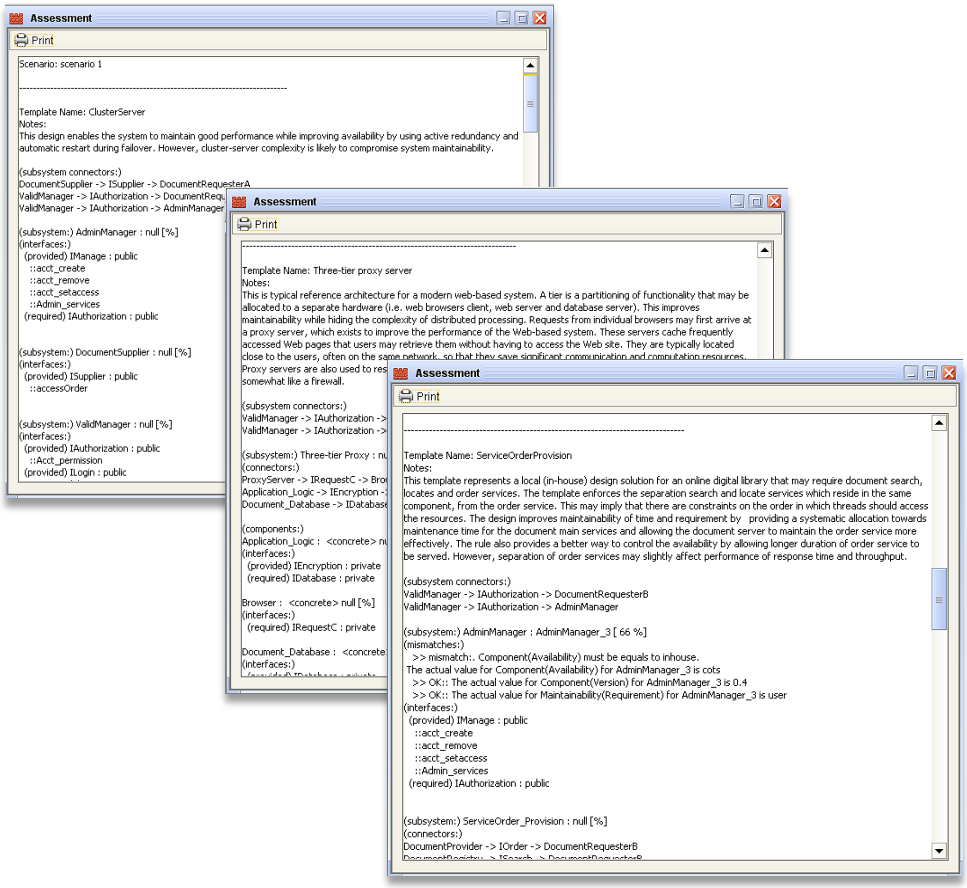


Fig. D6.1 *ServiceOrder Provision* reports

Table D6.1 Full reports extracted from report pane for Scenario 1

Scenario: scenario 1

Template Name: ClusterServer
Notes:
This design enables the system to maintain good performance while improving availability by using active redundancy and automatic restart during failover. However, cluster-server complexity is likely to compromise system maintainability.

```
(subsystem connectors:)
DocumentSupplier -> ISupplier -> DocumentRequesterA
ValidManager -> IAuthorization -> DocumentRequesterA
ValidManager -> IAuthorization -> AdminManager

(subsystem:) AdminManager : null [%]
(interfaces:)
  (provided) IManage : public
    ::acct_create
    ::acct_remove
    ::acct_setaccess
    ::Admin_services
  (required) IAuthorization : public


(subsystem:) DocumentSupplier : null [%]
(interfaces:)
  (provided) ISupplier : public
    ::accessOrder


(subsystem:) ValidManager : null [%]
(interfaces:)
  (provided) IAuthorization : public
    ::Acct_permission
  (provided) ILogin : public
    ::User_validation


(subsystem:) _ClusterServer : null [%]
(connectors:)
ClusterServer -> IDiscovery -> DocumentRequesterA
ClusterServer -> IRegistry -> DocumentRequesterA
DocumentRegistry1 -> IAccess -> ClusterServer
DocumentRegistry...n -> IAccess2 -> ClusterServer

(components:)
ClusterServer :  <concrete> null [%]
(interfaces:)
  (provided) IDiscovery : private
    ::accessSearch
  (provided) IRegistry : private
    ::accessLocate
  (required) IAccess : private
  (required) IAccess2 : private

DocumentRegistry...n :  <concrete> null [%]
(interfaces:)
  (provided) IAccess2 : private
    ::accessLocate
    ::accessSearch

DocumentRegistry1 :  <concrete> null [%]
(interfaces:)
  (provided) IAccess : private
    ::accessLocate
    ::accessSearch

DocumentRequesterA :  <concrete> null [%]
(interfaces:)
  (provided) IRequestA : public
    ::document_locate
```

```
   ::document_search
  (required) IAuthorization : public
  (required) IDiscovery : private
  (required) IRegistry : private
  (required) ISupplier : public

--------------------------------------------------------------------------------

Template Name: ServiceOrderProvision
Notes:
This template represents a local (in-house) design solution for an online digital library that may require
document search, locates and order services. The template enforces the separation search and locate services
which reside in the same component, from the order service. This may imply that there are constraints on the
order in which threads should access the resources. The design improves maintainability of time and
requirement by   providing a systematic allocation towards maintenance time for the document main services
and allowing the document server to maintain the order service more effectively. The rule also provides a better
way to control the availability by allowing longer duration of order service to be served. However, separation of
order services may slightly affect performance of response time and throughput.

(subsystem connectors:)
ValidManager -> IAuthorization -> DocumentRequesterB
ValidManager -> IAuthorization -> AdminManager

(subsystem:) AdminManager : AdminManager_3 [ 66 %]
(mismatches:)
   >> mismatch:. Component(Availability) must be equals to inhouse.
 The actual value for Component(Availability) for AdminManager_3 is cots
   >> OK:: The actual value for Component(Version) for AdminManager_3 is 0.4
   >> OK:: The actual value for Maintainability(Requirement) for AdminManager_3 is user
(interfaces:)
  (provided) IManage : public
    ::acct_create
    ::acct_remove
    ::acct_setaccess
    ::Admin_services
  (required) IAuthorization : public


(subsystem:) ServiceOrder_Provision : null [%]
(connectors:)
DocumentProvider -> IOrder -> DocumentRequesterB
DocumentRegistry -> ISearch -> DocumentRequesterB
DocumentRegistry -> ILocate -> DocumentRequesterB

(components:)
DocumentProvider : <concrete>DocumentSupplier_2 [ 83 %]
(mismatches:)
   >> OK:: The actual value for Component(Standard) for DocumentSupplier_2 is Z39.50
   >> OK:: The actual value for Component(Availability) for DocumentSupplier_2 is web service
   >> OK:: The actual value for Component(Cost) for DocumentSupplier_2 is 500 GBP
   >> OK:: The actual value for Component(Certification) for DocumentSupplier_2 is yes
   >> OK:: The actual value for Reliability(Availability) for DocumentSupplier_2 is 85 %
(interfaces:)
  (provided) IOrder : private
    ::accessOrder

DocumentRegistry : <concrete>DocumentRegistry1_1 [ 80 %]
(mismatches:)
   >> mismatch:: Component(Certification) must be equals to yes.
 The actual value for Component(Certification) for DocumentRegistry1_1 is no
   >> OK:: The actual value for Component(Cost) for DocumentRegistry1_1 is 400 GBP
   >> OK:: The actual value for Component(Standard) for DocumentRegistry1_1 is Z39.50
```

```
  >> OK:: The actual value for Component(Availability) for DocumentRegistry1_1 is web service
  >> mismatch:: Component(Certification) must be equals to yes.
 The actual value for Component(Certification) for DocumentRegistry1_1 is no
  >> OK:: The actual value for Component(Standard) for DocumentRegistry1_1 is Z39.50
  >> OK:: The actual value for Component(Availability) for DocumentRegistry1_1 is web service
  >> OK:: The actual value for Component(Cost) for DocumentRegistry1_1 is 400 GBP
  >> OK:: The actual value for Reliability(Availability) for DocumentRegistry1_1 is 75 %
  >> OK:: The actual value for Reliability(Availability) for DocumentRegistry1_1 is 75 %
(interfaces:)
 (provided) ILocate : private
  ::accessLocate
 (provided) ISearch : private
  ::accessSearch

DocumentRequesterB :  <concrete> null [%]
(mismatches:)
  >> mismatch:: Component(Certification) must be equals to yes.
 The actual value for Component(Certification) for DocumentRegistry1_1 is no
  >> OK:: The actual value for Component(Cost) for DocumentRegistry1_1 is 400 GBP
  >> OK:: The actual value for Component(Standard) for DocumentRegistry1_1 is Z39.50
  >> OK:: The actual value for Component(Availability) for DocumentRegistry1_1 is web service
  >> mismatch:: Component(Certification) must be equals to yes.
 The actual value for Component(Certification) for DocumentRegistry1_1 is no
  >> OK:: The actual value for Component(Standard) for DocumentRegistry1_1 is Z39.50
  >> OK:: The actual value for Component(Availability) for DocumentRegistry1_1 is web service
  >> OK:: The actual value for Component(Cost) for DocumentRegistry1_1 is 400 GBP
  >> OK:: The actual value for Reliability(Availability) for DocumentRegistry1_1 is 75 %
  >> OK:: The actual value for Reliability(Availability) for DocumentRegistry1_1 is 75 %
(interfaces:)
 (provided) IRequestB : public
  ::document_locate
  ::document_order
  ::document_search
  ::document_services
 (required) IAuthorization : public
 (required) ILocate : private
 (required) IOrder : private
 (required) ISearch : private


(subsystem:) ValidManager : null [%]
(interfaces:)
 (provided) IAuthorization : public
  ::Acct_permission
 (provided) ILogin : public
  ::User_validation

--------------------------------------------------------------------------------

Template Name: Three-tier proxy server
Notes:
This is typical reference architecture for a modern web-based system. A tier is a partitioning of functionality that
may be allocated to a separate hardware (i.e. web browsers client, web server and database server). This
improves maintainability while hiding the complexity of distributed processing. Requests from individual
browsers may first arrive at a proxy server, which exists to improve the performance of the Web-based system.
These servers cache frequently accessed Web pages that users may retrieve them without having to access the
Web site. They are typically located close to the users, often on the same network, so that they save significant
communication and computation resources. Proxy servers are also used to restrict users' access to certain Web
sites. In this case the proxy server is acting somewhat like a firewall.

(subsystem connectors:)
ValidManager -> IAuthorization -> AdminManager
```

```
ValidManager -> IAuthorization -> DocManager

(subsystem:) Three-tier Proxy : null [%]
(connectors:)
ProxyServer -> IRequestC -> Browser
Application_Logic -> IEncryption -> ProxyServer
Document_Database -> IDatabase -> Application_Logic

(components:)
Application_Logic :  <concrete> null [%]
(interfaces:)
  (provided) IEncryption : private
  (required) IDatabase : private

Browser :  <concrete> null [%]
(interfaces:)
  (required) IRequestC : private

Document_Database :  <concrete> null [%]
(interfaces:)
  (provided) IDatabase : private
    ::accessLocate
    ::accessOrder
    ::accessSearch

ProxyServer :  <concrete> null [%]
(interfaces:)
  (provided) IRequestC : private
    ::document_services
  (required) IEncryption : private
  (required) ILogin : private
  (required) IManage : private
  (required) IQuery : private

--------------------------------------------------------------------------------
```

Appendix E:

GVPS Detail Specifications & Results

E1. GVPS Software Requirements Specification

Software Requirements Specification

for

Guided Vehicle Parking System

Version 1.0

Prepared by Novia Admodisastro

Lancaster University

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

1.1 Purpose

This document describes detailed requirement specification for a Guided Vehicle Parking System (GVPS) for Lancaster University. The objective is to provide drivers with accurate and timely parking information and thus reduce their travel times while decreasing number of cars searching for empty spaces.

The GPVS consist of two main components which are In-Car Display (ICD) and Control Centre.

ICD allow a user driving a car to enter the university campus, and simply be selecting a destination on their in-car screen, be assigned to the best available space on the closest possible car park to their destination. When leaving the university, the system, will also provide directions and back to the exit.

ICD communicate with the central server and display a map of the campus roads and car parks, highlighting the route to be taken and showing the correct direction to be taken at junctions and roundabouts both visually and audio. ICD also informs users of different traffics messages according to its locations and distance to an incident, such as traffic signs, and road-works and indicates alternative routes as appropriate.

The system also provides a Control Centre for administrative users who can monitor the status of each vehicle and car park on campus, and enable closure of sections of road in case of emergency or maintenance.

1.2 Intended Audience and Reading Suggestions

This is a technical document meant for system developers and users.

1.3 Product Scope

As Lancaster University is piloting the GVPS for other UK institutions, it must be designed to read in common traffic and structural details found on a university campus including road layout, buildings, and parking spaces. Appendix 1 provides Lancaster University parking specification.

2. Overall Description

2.1 Operating Environment

The system shall operate on the Windows XP platform. It requires Microsoft SQL Server 2000 to host the database tables and a Ethernet LAN for client-server access to the database server.

2.2 Design and Implementation Constraints

Because of the existing windows platform and forthcoming Windows Local Area Network (LAN), the system shall be developed using Component-Based Software Development (CBSE) approach using JavaBeans component model. This will leverage existing expertise held by the developers. The database tables will be hosted on Microsoft SQL Server 2000 and accessed via JDBC.

2.3 Assumptions and Dependencies

For the system to run effectively, a database server must be installed on a server machine. This server machine must be accessible across a LAN. We are assuming that we will have a LAN running and server machine to host the database.

2.4 Priority of Requirements

Within in this document, all requirements are categorised under three priority levels:

- ESSENTIAL (3): This means that the requirement is crucial for all GVPS components (IVD, Control Centre and/or Simulator), if they are to adequately deliver commitments made on them by operators and stakeholders.
- IMPORTANT (2): This means that the requirement may prove extremely useful in assisting GVPS in delivering their commitments i.e. reducing the amount effort required by the organisation's staff by increasing the level of automation.
- USEFUL (1): This means the requirement could prove useful in processing IVD, Control Centre and/or Simulator requests, but it is far more likely to only be of use to a subset of GVPS operators and stakeholders.

2.5 Abbreviations

Abbreviations	Description
GVPS	Guided Vehicle Parking System
LAN	Local Area Network
CBSE	Component-Based Software Development
Prio.	Priority
RFID	Radio Frequency Identification Tags

3. Requirements

3.1 Vehicle Requirements

- Access to the campus wireless network
- Onboard computer with small LCD display, input device and voice synthesis capability.
- An onboard GPS system with accuracy of 2m. (can be read by onboard computer).
- Odometer with accuracy better than 1m in 50m. (can be read by onboard computer).

3.2 GVPS Requirements

3.2.1 ICD

- ***Connect***; To enable drivers either holding a car permit or visitor to access GVPS 24/7.
- ***Login***; The GVPS will automatically identify the vehicle by means of a RFID tag embedded in the campus parking permits, or display identification page for visitor.
  - Permit holder: Earlier registration require them to provide detail as the following:
    1. Driver name
    2. Vehicle registration no.
    3. Vehicle type (i.e. Car – C, Disabled – D, or Van/Lorry – O)
    4. Permit type (i.e. Staff – F or Student – S)

    They will be assigned:
    1. RFID (auto generate ID)
  - Visitor (V): Vehicle without permit will automatically identify as a visitor vehicle and will be assigned a temporary ID.
    1. Vehicle registration no.

2. Vehicle type (i.e. CV, DV, or OD)

- ***Destination***; To enable drivers to enter their campus destination at an entrance and calculate provide routing according to shortest route.
- ***Guide***; To guide the driver of the vehicle to a designated parking place (given as a particular car park) as close to the destination as possible, taking into account the type of vehicles (C, D, or O) and driver categories (F, S or V).
- ***Exit***; To guide the driver of the vehicle to an exit.
- Traffic messages; To informs drivers of different traffics messages according to its locations and distance to an incident, such as traffic signs (refer Appendix A), and road-works and indicates alternative routes as appropriate.
- ***Wrong turning***; To informs drivers of wrong turning is made and re-calculate route.
- ***Display***; The display in the vehicle will show the position of the vehicle on a map and provide timely guidance to direct the driver to the designated parking space. The GUI will also allow the user to interact as deemed necessary.

3.2.2 Control Centre

- ***Login***; To enable admin to access GVPS 24/7 in a secure environment.
- ***Driver accounts***; To manage driver accounts.
- ***Map***; To manage map including to parse the map.
- ***Vehicle status***; To monitor the status of all vehicles that accessing GVPS.
- ***Car parks***; To monitor car parks on campus.
- ***Road closure***; To enable closure of sections of road in case of emergency or maintenance.
- ***Display***; To provide visualization tracking for all vehicle accessing GVPS.

3.3 Non-Functional Requirements

The main non-functional categories associated with the system(s) include:

- Efficiency; GVPS shall efficiently manage to scarce computational resources (i.e. CPU cycles and memory) to handle high consumptions tasks e.g. drawing and displaying map, and monitoring vehicles.
- Performance; GVPS shall provides reasonable level of performance IVD and control centre to receiving and sending data.
- Reliability; GVPS shall allow driver for 24/7 access.
- Security; GVPS shall provides a secure environment for admin and control privacy.
- Flexibility; GVPS shall be expendable for any other UK institutions.
- Documentation / Online help;
- Training;

3.4 Summary

The GVPS requirements are summarize as in Table 3.1. These requirements represent viewpoints of GVPS operators, component and stakeholder.

1. GVPS Operators:
   - ***Driver***; Driver is a person whom interested to access In Vehicle Display (IVD). IVD helps them to navigate the road campus to find a car park and to exit campus. Driver is either member of Lancaster University or visitor.
     Driver type member is require to register his/her vehicle to the GVPS management to ensure appropriate parking space (based on permit type) is been assigned.
     Unrecognised vehicle is assume as Driver type visitor and being assign a temporary ID which only allow him/her for visitor parking space. However, visitor can indicate disability requirement.
   - ***Admin***; Admin is a person whom responsible to manage GVPS system such as vehicle handling, parking space handling, road obstruction handling, assigning car park to driver and able to view all vehicle connecting GVPS.
2. GVPS Components:
   - ***Traffics***; Existing component that consist of dynamic traffic signal from campus such as traffic lights and speed detector.
3. GVPS Stakeholders:
   - ***Consortium***; Consortium is Lancaster University whom invested in GVPS and wanted the system to be delivered in certain duration of time. The consortium may represent domain characteristic and constraints that may influence the system requirements.

Table 3.1. GVPS requirements summary

Viewpoint		**Requirement**			
ID	**Role/Type**	**ID**	**Description**	**Rationale**	**Ranking**
Vp_1	GVPS_Driver (Operator)	R1.1	To enable drivers either holding a car permit or visitor to access GVPS 24/7.	To provide connection to GVPS server.	2
		R1.2	To be able to logon to the system using valid RFID or registration number.	To provide an access to IVD services.	3
		R1.3	To guide the driver of the vehicle to a designated parking place (given as a particular car park) as close to the destination as possible.	Basic IVD functionality	3
		R1.4	The display in vehicle will show the position of the vehicle on a map.	Basic IVD functionality	3
		R1.5	To guide the driver of the vehicle to an exit.	Basic IVD functionality	3
		R1.6	To informs drivers of different traffics messages according to its locations and distance to an incident	To provide an alert to the driver.	2
		R1.7	To informs drivers of wrong	To guide driver on	3

			turning is made and re-calculate route	track to destination.	
Vp_2	GVPS_ Administrator (Operator)	R2.1	To enable admin to access GVPS 24/7 in a secure environment.	To provide an access to Control Centre services.	3
		R2.2	To manage driver accounts i.e. add/delete/update accounts.	To provide a mechanism to manage registered vehicles.	3
		R2.3	To manage map i.e. add/delete map.	To provide a mechanism to manage map.	3
		R2.4	To manage car parks on campus by providing the status of car parks.	To provide a mechanism to monitor the status of car parks in campus	2
		R2.5	To enable closure of sections of road in case of emergency or maintenance.	To provide a notice for any road obstruction	2
		R2.6	To monitor the status of all vehicles accessing GVPS.	To provide a mechanism to track status of all vehicle.	3
Vp_3	GVPS_ Traffics (Component)	R3.1	The IVD client will act as an observer for traffics signal broadcast.	To provide real time traffics message to the driver.	2
Vp_4	GVPS_ Consortium (Organisation)	R4.1	The system shall efficiently manage scarce computational resources (i.e. CPU cycles and memory).	GVPS shall manage resources for high consumptions tasks (i.e. drawing and displaying map, and monitoring vehicles) systematically.	3
		R4.2	The system shall ensure a reasonable level of performance is maintained across the services at all times.	GVPS shall provide reasonable level of performance IVD and control centre to receiving and sending data.	3
		R4.3	The system shall provide 24/7 access.	GVPS shall be accessible by driver to find car parking and admin to monitor parking areas.	1
		R4.4	The system shall enforce authentication policies to avoid loss of data integrity or confidentiality	The network should provide a secure environment and control privacy.	3
		R4.5	The system shall promote XML data map format and driver independence on map resources.	GVPS shall be flexible and reusable to be adopted for any other UK institutions GVPS map shall be easily interpreted and upgraded to be adopted for any other UK institutions.	2
		R4.6	The system shall be develop according to schedule and cost estimated.		1

4. Service Descriptions

GVPS service descriptions are shown in Table 4.1 – 4.13.

Table 4.1 SetupConnection service description

GVPS: S1.1.1 SetupConnection	
Actors	GVPS_Driver
Description	1. GVPS driver find connection 2. If connection is valid then 2.1 If map not exist 2.1.1 GVPS driver load new map and assign map ID 2.1.2 System initialise server connection 2.2 else if map exist 2.2.1 Choose and load map 2.2.2 System initialise server connection 2.3 else 2.3.1 prompt an error message – connection cannot be established
Entry conditions	1. Valid connection 2. Valid vehicle registration
Exit conditions	1. System reset vehicle 2. Closes server connection
Constraints	1. The service shall be available on Microsoft Mobile platform 2. Service shall have a reasonable level of performance at all times 3. The service shall have 24/7 access availability.

Table 4.2 LoginIVD service description

GVPS: S1.2.1 LoginIVD; S1.2.2 Validate	
Actors	GVPS_Driver
Description	1. System prompt vehicle registration page 2. If RFID tag is valid 2.1 System initialise driver system permissions 2.2 Display the services available to the driver else 2.3 Driver enter vehicle registration no and type (C-V, D-V, or O-V) 2.4 System assigned temporary ID else 2.5 System prompts an error message
Entry conditions	1. Valid RFID or Vehicle Registration No.
Exit conditions	1. System access conditions is reset 2. Closes user account.
Constraints	1. RFID is conforms to standard passive tag protocol. 2. The service shall be easy to maintain to current technology and requirements.

Table 4.3 SearchParking service description

GVPS: S1.3.1 SearchParking; S1.3.2 Request Parking	
Actors	GVPS_Driver
Description	1. GVPS driver enters destination (building/department) name 2. If destination is found then 2.1 If vehicle is type D then 2.1.1 A nearest parking space to the destination D type will be reserved else if vehicle is type C and eligibility is F 2.1.2 A nearest parking space to the destination C type with eligibility F will be reserved else if vehicle is type C and eligibility is S 2.1.3 A nearest parking space to the destination type C with S eligibility will be reserve

	else if vehicle is type C and eligibility is V 2.1.4 A nearest parking space to the destination type C with V eligibility will be reserve else if vehicle is type O and eligibility is F 2.1.5 A nearest parking space to the destination type O with F eligibility will be reserved else if vehicle is type O and eligibility is S 2.1.6 A nearest parking space to the destination type O with S eligibility will be reserved else if vehicle is type O and eligibility is V 2.1.1 A nearest parking space to the destination type O with V eligibility will be reserved 3. Shortest path to the destination will be calculated 4. Car park is reserved until users release destination.
Entry conditions	1. Vehicle_RFID ∈available_RFID 2. Map databases ⊆ set of user permissible databases
Exit conditions	1. System access conditions is reset
Constraints	1. Car space allocation is restricted with vehicle type, eligibility and availability. 2. Shortest path taking into consideration of route alternative when road obstruction occurs.

Table 4.4 NavigateRoute service description

GVPS: S1.4.1 NavigateRoute	
Actors	GVPS_Driver
Description	1. If map found, 1.1 Map and its entities is display else 1.2 prompt error message “map not found!” 2. Display vehicle location in a map 3. Re-draw vehicle movements on the map.
Entry conditions	1. Vehicle_RFID ∈available_RFID 2. Map databases ⊆ set of user permissible databases
Exit conditions	1. System access conditions is reset
Constraints	1. Map is given in coordinate (x,y) in txt file.

Table 4.5 Exit service description

GVPS: S1.5.1 Exit	
Actors	GVPS_Driver
Include	IVD_Console
Description	1. GVPS driver selects exit south or north. 2. If exit destination is valid 2.1 Shortest path to the exit will be calculated. 2.2 Car space is release. else 2.3 An error message “invalid exit” is display.
Entry conditions	1. Vehicle_RFID ∈available_RFID 2. Map databases ⊆ set of user permissible databases
Exit conditions	1. System access conditions is reset
Constraints	1. Service conforms to communication server protocol standard.

Table 4.6 Traffic service description

GVPS: S.1.6.1 BroadcastTrafficSignal	
Actors	Traffic, GVPS_Driver
Description	1. System observe signal from Traffic or ControlCentre. 2. If signal message from Traffic is valid 2.1 Checked traffic signs message 2.2 Broadcast appropriate message to GVPS Driver. else 2.3 An error message “invalid signal” is display.

Entry conditions	1. Vehicle_RFID ∈available_RFID 2. Map databases ⊆ set of user permissible databases
Exit conditions	2. System access conditions is reset
Constraints	3. The service shall have a reasonable level of performance of broadcasting traffic messages. 4. The service shall able to read standard signal broadcast by Traffic.

Table 4.7 WrongTurning service description

GVPS: S1.7.1 WrongTurning	
Actors	GVPS_Driver
Description	1. If system navigation not equals to destination 1.1 Prompt message “wrong route” 1.2 Re-calculate shortest path from location to destination 2. Prompt new routing to the GVPS driver.
Entry conditions	1. Vehicle_RFID ∈available_RFID 2. Map databases ⊆ set of user permissible databases
Exit conditions	1. System access conditions is reset
Constraints	-

Table 4.8 Login service description

GVPS: S2.1.1 Login; S2.1.2 ValidateAdmin	
Actors	GVPS_Administrator
Description	3. GVPS admin request for login 4. GVPS admin enters a username and password 5. Verify login against a set of username-password pairs in the database 6. If username and password are valid: 4.1 System initialise user account permissions 4.2 Display the services available to the user else 4.3 System prompts the user to re-enter username and password with three attempts.
Entry conditions	2. Valid username 3. Valid password
Exit conditions	3. System access conditions is reset 4. Closes user account.
Constraints	1. The service shall provide a reasonable level of security. 2. The service shall be easy to maintain to current technology and requirements.

Table 4.9 ManageDriver service description

GVPS: S2.2.1 ManageDriver	
Actors	GVPS_Administrator, GVPS_Member
Description	1. GVPS admin manage for driver account 2. If add driver then system request driver: Driver name – provide char(25) Car plat number – provide alphanumneric(7) Parking permit type – select from combo box Vehicle permit – select from combo box 3. If remove driver 3.1 system request search driver 4. If driver is found 4.1 delete driver details from database 5. If search driver then 5.1 system request driver name or car plat number 6. If driver is found 6.2 driver details is displayed else 6.2 display message “driver not found”

Entry conditions	1. Manage_map ∈available_services 2. Vehicle databases ⊆ set of permissible databases
Exit conditions	1. System access conditions is reset
Constraints	1. Parking permit type A or B 2. Vehicle permit type C, D, or O

Table 4.10 ManageMap service description

GVPS: S2.3.1 ManageMap; S2.3.2 ParseMap; S2.3.3 Map Entities	
Actors	GVPS_Administrator
Description	1. GVPS admin search for campus map. 2. GVPS admin browse map directory 3. If map is found 3.1 System parse map entities and display map else 3.2 error message displayed “Map not found” 4. Map directory is retained in user workspace for future locates
Entry conditions	1. Manage_map ∈available_services 2. Vehicle databases ⊆ set of permissible databases
Exit conditions	1. System access conditions is reset
Constraints	1. Map entities is in *.txt file. 2. The service should allow flexibility and reusability to parse different type of map entities.

Table 4.11 ManageParking service description

GVPS: S2.4.1 ManageParking	
Actors	GVPS_Administrator
Description	1. GVPS admin monitor parking status in university campus 2. If a vehicle assigned to a car park, 2.1 parking space is reduce to 1 3. If a vehicle exit from a car park, 3.1 parking space is increase to 1 4. Car parks and its parking spaces are retained in working space.
Entry conditions	1. Manage_map ∈available_services 2. Vehicle databases ⊆ set of permissible databases
Exit conditions	1. System access conditions is reset
Constraints	-

Table 4.12 ManageObstruction service description

GVPS: S2.5.1 ManageObstruction	
Actors	GVPS_Administrator
Description	1. GVPS admin manage road obstruction in university campus. 2. If obstruction 2.1 GVPS admin enters a set of obstructed road segments and notify IVD 3. If obstruction resolved 3.1 GVPS admin remove the road segments and notify IVD.
Entry conditions	1. Manage_map ∈available_services 2. Vehicle databases ⊆ set of permissible databases
Exit conditions	1. System access conditions is reset
Constraints	-

Table 4.13 ViewVehicleStatus service description

GVPS: S2.6.1 ViewVehicleStatus; S2.6.2 VehicleTracker; S2.6.3 TrafficTracker	
Actors	GVPS_Administrator
Description	1. If map found, 1.1 Map and its entities is display else 1.2 Prompt error message “map not found!”

	2. Display vehicles location in a map 3. Re-draw each vehicle movements on the map.
Entry conditions	1. Manage_map ∈available_services 2. Vehicle databases ⊂ set of permissible databases
Exit conditions	1. System access conditions is reset
Constraints	1. Map is given in coordinate (x,y) in txt file. 2. The service shall be effectively managed.

5. Constraint Descriptions

GVPS constraint descriptions are shown in Table 5.1.

Table 5.1 GVPS constraint descriptions

Concern	Sub-concern	ID	Description	Rationale	Scope
1-Efficiency	1-Memory	C1.1.1	The system shall efficiently manage consumptions of memory.	To ensure bottle neck and effective resource management.	S2.6.2
	2-Processor	C1.2.1	The system shall efficiently manage high consumptions of processor.	To ensure bottle neck and effective resource management.	S2.6.2
2-Performance	1-Response time	C.2.1.1	GVPS shall allow provide reasonable level of performance IVD and Control Centre to receive and send data	To ensure that a reasonable level of performance is given to drivers and admin	System
3-Reliability	1-Availability	C3.1.1	GVPS shall allow driver for 24/7 access.	To ensure sufficient GVPS access time.	S1.1.1
4-Security	1-Integrity	C4.1.1	GVPS shall provide a secure environment for admin access.	Secure control centre for admin	S2.1.1
		C4.1.2	GVPS shall provide a secure environment for control privacy	Secure RFID for drivers	S1.2.1
5-Flexibility	1-Expendability	C5.1.1	GVPS shall be flexible to be adopted for any other UK institutions	Must be able to read common traffic and map structural details.	S2.3.1
6-Business	1-Cost	C6.1.1	System development according to cost estimated	Under medium budget	System
	2-Schedule	C6.2.1	System development according to schedule estimated.	Under normal delivery date	System
	3-Platform	C6.3.1	The system shall run on Microsoft Windows Mobile	Sufficient support for intended purpose	S1.2.1 S1.3.1 S1.5.1 S1.7.1
	4-Component Model	C6.4.1	The system shall develop using JavaBeans component model	Leverage existing expertise held by the developers.	System

7-Component	1-Availability	C7.1.1	GVPS_Traffic subscribe to available traffics components.	Using existing component in navigation system	S1.6.1
	2-Standard	C7.2.1	RFID is conform to standard passive tag control	Vehicle navigation standard for accessing IVD.	S1.2.1
	3-Persistent	C7.3.1	The database hosted on Ms. SQL Server 2000 accessed via JDBC	Sufficient support for intended purpose	S2.2.1 S2.3.1

Appendix 1: Lancaster University Car Parking Specification

- Car parks monitored: 23
- Parking spaces monitored: 500
- Type of vehicles:
  - Car (C)
  - Car-disabled (D)
  - Van/Lorry (O)
- Permits and eligibility:
  - Staff (F)
  - Students (S)
  - Others: Visitor (V) – All visitor's vehicle (that is any vehicle not displaying a staff, students or contractor parking permit) will require either a pay and display ticket when parked on campus.
- Car parks locations:
  - Alexandra Park Drive 30 spaces
  - Bowland Ave. 15 spaces
  - Bowland Avenue 28 spaces
  - Cartmel West Ave. 15 spaces
  - Farrer Avenue 72 spaces
  - Fylde Ave 15 spaces
  - Gillow Ave. 15 spaces
  - Graduate North Ave. 20 spaces
  - John Creed Ave. 10 spaces
  - Library Ave. 15 spaces
  - Lonsdale South Ave. 20 spaces
  - Management School 12 spaces
  - North Drive 20 spaces
  - North East Drive 20 spaces
  - North West Drive 20 spaces
  - Physics Ave. 20 spaces
  - Rossendale Ave. 20 spaces
  - South Bowland Ave East 15 spaces
  - South Drive 20 spaces
  - South Drive 56 spaces
  - South East Drive 36 spaces
  - Tower Ave. 10 spaces
  - Whewell Building 6 spaces
- Visitor car parks are in the following locations:
  - Farrer Avenue 72 spaces
  - South Drive 56 spaces
  - Bowland Avenue 28 spaces
  - Management School 12 spaces
  - Whewell Building 6 spaces
- The map entities:
  - Car parks and spaces as describe above.
  - Roads: 56 roads segments
  - Traffic signs
    i) Traffic lights
    ii) Pedestrian
    iii) Bus stops

- Entrances:
  i) A6 North Entrance
- Buildings:

Ash House
Biological and Environmental Science
Bowland Annex
Bowland College
Bowland Hall South
Bowland Hall North
Bowland Lecture Theatre
Bowland North
Bowland Tower
Bowland Tower East
Bowland Tower South
Central Workshops and Stores
CETAD
Chaplaincy Centre
Computer Services
Conference Centre
County College
County South
County West
Engineering Building
Faraday Building
Former Cartmell College
Former County College
Furness College
Furness Residences
Fylde College
Fylde Residences
George Fox Building
Great Hall and Peter Scott Gallery
Grizedale College
Health Centre
InfoLab21
Jack Hylton Music Rooms
John Creed Buiding
Lancaster Environment Centre
LEC Workshops
Library
LUTV - Round House
Management School
Nuffield Theatre
Pendle Bar
Pendle College
Physics Building
Post Office
Pre-school Centre
Reception Building
Ruskin Library
Slaidburn House
Sports Centre
University House
Whewell Building

Campus map is as Fig. A1.1:

Fig. A1.1. Lancaster campus map

E2. iXML ADL Specification of GVPS

GVPS architecture detailed specification is shown in Table E2.1.

Table E2.1. iXML ADL Specification of GVPS

```
<?xml version = '1.0' encoding = 'UTF-8' ?>
<!DOCTYPE iXML SYSTEM "/Documents and Settings/norwy/NetBeans_projects/csafe/iXML.dtd">
<iXML>
<component name.id = 'CC_Console' type = '' visibility = 'private'>
<component.description>
CC_Console component is for administrative users who can monitor the status of each vehicle and car park on campus, and enable closure of sections of road in case of emergency or maintenance.
</component.description>
<component.interface  name.idref = 'IDataCentre' port.idref = 'r'/>
<component.interface  name.idref = 'IMapCC' port.idref = 'r'/>
<component.interface  name.idref = 'IControlCentre' port.idref = 'p'/>
<component.interface  name.idref = 'IRouteObs' port.idref = 'p'/>
<component.connector name.idref = 'IDataCentre -> CC_Console'/>
<component.connector name.idref = 'IMapCC -> CC_Console'/>
<component.constraint concern = 'Security' subconcern = 'Integrity' type = 'invariant' state = 'EL' value = 'authentication_policies' scope = 'Login'/>
<component.constraint concern = 'Component' subconcern = 'Persistent' type = 'precondition' state = 'EL' value = 'SQL Server' scope = 'ManageDriver'/>
<component.constraint concern = 'Component' subconcern = 'Persistent' type = 'precondition' state = 'EL' value = 'SQL Server' scope = 'ManageMap'/>
<component.property concern = 'Component' subconcern = 'Availability' value = 'inhouse'/>
</component>
<component name.id = 'IVD_Console' type = '' visibility = 'private'>
<component.description>
ICD_Console component provides services to the driver to navigate campus to and from parking space using a valid registration number or RFID tag.
</component.description>
<component.interface  name.idref = 'IVehicle' port.idref = 'p'/>
<component.interface  name.idref = 'IDataIVD' port.idref = 'r'/>
<component.interface  name.idref = 'IMapIVD' port.idref = 'r'/>
<component.interface  name.idref = 'IRouteObs' port.idref = 'r'/>
<component.connector name.idref = 'IRouteObs -> IVD_Console'/>
<component.connector name.idref = 'IDataIVD -> IVD_Console'/>
<component.connector name.idref = 'IMapIVD -> IVD_Console'/>
<component.constraint concern = 'Component' subconcern = 'Standard' type = 'precondition' state = 'EL' value = 'PassiveTag' scope = 'LoginIVD'/>
<component.constraint concern = 'Security' subconcern = 'Integrity' type = 'invariant' state = 'EL' value = 'authentication_policies' scope = 'LoginIVD'/>
<component.constraint concern = 'Reliability' subconcern = 'Availability' type = 'precondition' state = 'GT' value = '100' scope = 'SetupConn'/>
<component.constraint concern = 'Business' subconcern = 'Platform' type = 'invariant' state = 'EL' value
```

```
= 'Windows Mobile' scope = 'LoginIVD'/>
<component.constraint concern = 'Business' subconcern = 'Platform' type = 'invariant' state = 'EL' value
= 'Windows Mobile' scope = 'SearchParking'/>
<component.constraint concern = 'Business' subconcern = 'Platform' type = 'invariant' state = 'EL' value
= 'Windows Mobile' scope = 'Exit'/>
<component.constraint concern = 'Business' subconcern = 'Platform' type = 'invariant' state = 'EL' value
= 'Windows Mobile' scope = 'WrongTurning'/>
<component.property concern = 'Component' subconcern = 'Standard' value = ' PassiveTag'/>
<component.property concern = 'Component' subconcern = 'Availability' value = 'Inhouse'/>
<component.property concern = 'Business' subconcern = 'Platform' value = 'Windows Mobile'/>
</component>
<component name.id = 'avpsDB' type = '' visibility = 'private'>
<component.description>
The component provides persistent storage of driver, vehicle and campus entities.
</component.description>
<component.interface  name.idref = 'IDataCentre' port.idref = 'p'/>
<component.interface  name.idref = 'IDataIVD' port.idref = 'p'/>
<component.interface  name.idref = 'IDataMap' port.idref = 'p'/>
<component.property concern = 'Component' subconcern = 'Persistent' value = 'SQL Server'/>
<component.property concern = 'Component' subconcern = 'Availability' value = 'inhouse'/>
</component>
<component name.id = 'Map' type = '' visibility = 'private'>
<component.description>
The component provides services to visualise campus entities including traffic signal
</component.description>
<component.interface  name.idref = 'IMapIVD' port.idref = 'p'/>
<component.interface  name.idref = 'IMapCC' port.idref = 'p'/>
<component.interface  name.idref = 'IDataMap' port.idref = 'r'/>
<component.connector name.idref = 'IDataMap -> Map'/>
<component.constraint concern = 'Component' subconcern = 'Availability' type = 'invariant' state = 'EL'
value = ' inhouse' scope = 'TrafficSignal'/>
<component.constraint concern = 'Flexibility' subconcern = 'Expendability' type = 'invariant' state = 'EL'
value = 'xml-based' scope = 'DrawMap'/>
<component.constraint concern = 'Efficiency' subconcern = 'Memory' type = 'postcondition' state = 'LE'
value = '20' scope = ' VehicleTracker '/>
<component.constraint concern = 'Efficiency' subconcern = 'Processor' type = 'postcondition' state =
'EL' value = '75' scope = 'VehicleTracker'/>
<component.property concern = 'Component' subconcern = 'ComponentModel' value = 'JavaBeans'/>
</component>
<component name.id = 'Navi' type = 'subsystem' visibility = 'public'>
<component.description>Subsystem</component.description>
<component.composite name.idref = 'CC_Console'/>
<component.composite name.idref = 'IVD_Console'/>
<component.composite name.idref = 'avpsDB'/>
<component.composite name.idref = 'Map'/>
<component.constraint concern = 'Business' subconcern = 'Schedule' type = 'invariant' state = 'EL'
value = 'moderate' scope = 'System'/>
<component.constraint concern = 'Business' subconcern = 'Cost' type = 'invariant' state = 'EL' value =
'moderate' scope = 'System'/>
<component.constraint concern = 'Business' subconcern = 'ComponentModel' type = 'precondition'
```

```
state = 'EL' value = 'JavaBeans' scope = 'System'/>
<component.constraint concern = 'Performance' subconcern = 'ResponseTime_UPL' type = 'postcondition' state = 'LE' value = '0.5' scope = 'System'/>
<component.constraint concern = 'Performance' subconcern = 'ResponseTime_PL' type = 'postcondition' state = 'LE' value = '4' scope = 'System'/>
</component>
<interface name.id = 'IDataCentre' type = '' port = 'p' visibility = 'private'>
<interface.description>avpsDB provides this interface to CC_Console</interface.description>
<interface.service name = 'ValidateAdmin'>
<service.operation name.idref = 'validate'/>
</interface.service>
<interface.service name = 'ParseMap'>
<service.operation name.idref = 'parseMap'/>
</interface.service>
<interface.service name = 'ManageParking'>
<service.operation name.idref = 'queryDriver'/>
<service.operation name.idref = 'queryParking'/>
</interface.service>
<interface.operation name = 'parseMap' ret = ''>
<operation.param name = 'map' type ='String'/>
<operation.param name = 'entities' type ='vector'/>
</interface.operation>
<interface.operation name = 'validate' ret = ''>
<operation.param name = 'username' type ='String'/>
<operation.param name = 'pwd' type ='String'/>
</interface.operation>
<interface.operation name = 'queryDriver' ret = ''>
<operation.param name = 'driverID' type ='String'/>
<operation.param name = 'vehicleID' type ='String'/>
</interface.operation>
<interface.operation name = 'queryParking' ret = ''>
<operation.param name = 'parkAreaID' type ='integer'/>
</interface.operation>
</interface>
<interface name.id = 'IDataCentre' type = '' port = 'r' visibility = 'private'>
<interface.description>CC_Console requires this interface</interface.description>
<interface.operation name = 'parseMap' ret = ''>
<operation.param name = 'map' type ='String'/>
<operation.param name = 'entities' type ='vector'/>
</interface.operation>
<interface.operation name = 'validate' ret = ''>
<operation.param name = 'username' type ='String'/>
<operation.param name = 'pwd' type ='String'/>
</interface.operation>
<interface.operation name = 'queryDriver' ret = ''>
<operation.param name = 'driverID' type ='String'/>
<operation.param name = 'vehicleID' type ='String'/>
</interface.operation>
<interface.operation name = 'queryParking' ret = ''>
<operation.param name = 'parkAreaID' type ='integer'/>
```

```
</interface.operation>
</interface>
<interface name.id = 'IDataIVD' type = '' port = 'p' visibility = 'private'>
<interface.description>avpsDB provides this interface to IVD_Console</interface.description>
<interface.service name = 'ValidateDriver'>
<service.operation name.idref ='authenticate'/>
</interface.service>
<interface.service name = 'RequestParking'>
<service.operation name.idref ='queryParking'/>
</interface.service>
<interface.operation name = 'authenticate' ret = ''>
<operation.param name = 'vehicleNo' type ='String'/>
<operation.param name = 'rfidNo' type ='String'/>
</interface.operation>
<interface.operation name = 'queryParking' ret = 'integer'>
<operation.param name = 'dest' type ='String'/>
</interface.operation>
</interface>
<interface name.id = 'IDataIVD' type = '' port = 'r' visibility = 'private'>
<interface.description>IVD_Console requires this interface</interface.description>
<interface.operation name = 'authenticate' ret = ''>
<operation.param name = 'vehicleNo' type ='String'/>
<operation.param name = 'rfidNo' type ='String'/>
</interface.operation>
<interface.operation name = 'queryParking' ret = 'integer'>
<operation.param name = 'dest' type ='String'/>
</interface.operation>
</interface>
<interface name.id = 'IMapCC' type = '' port = 'p' visibility = 'private'>
<interface.description>Map provides this interface to CC_Console </interface.description>
<interface.service name = 'DrawMap'>
<service.operation name.idref = 'connectWaypoint'/>
</interface.service>
<interface.service name = 'TrafficTracker'>
<service.operation name.idref = 'trafficTracker'/>
</interface.service>
<interface.service name = 'VehicleTracker'>
<service.operation name.idref = 'showVehicle'/>
</interface.service>
<interface.operation name = 'showVehicle' ret = ''>
<operation.param name = 'vehicleID' type ='String'/>
</interface.operation>
<interface.operation name = 'connectWaypoint' ret = ''>
<operation.param name = 'coord' type ='String'/>
</interface.operation>
 <interface.operation name = 'trafficTracker' ret = ''>
</interface.operation>
</interface>
<interface name.id = 'IMapCC' type = '' port = 'r' visibility = 'private'>
<interface.description>CC_Console requires this interface</interface.description>
```

```
<interface.operation name = 'showVehicle' ret = ''>
<operation.param name = 'vehicleID' type ='String'/>
</interface.operation>
<interface.operation name = 'connectWaypoint' ret = ''>
<operation.param name = 'coord' type ='String'/>
</interface.operation>
<interface.operation name = 'trafficTracker' ret = ''>
</interface.operation>
</interface>
<interface name.id = 'IMapIVD' type = '' port = 'p' visibility = 'private'>
<interface.description>Map provides this interface to IVD_Console</interface.description>
<interface.service name = 'NavigateRoute'>
<service.operation name.idref = 'calculateRoute'/>
<service.operation name.idref = 'drawRoute'/>
</interface.service>
<interface.service name = 'TrafficSignal'>
<service.operation name.idref = 'traffiCast'/>
</interface.service>
<interface.operation name = 'calculateRoute' ret = ''>
<operation.param name = 'dest' type ='String'/>
</interface.operation>
<interface.operation name = 'drawRoute' ret = ''>
<operation.param name = 'coord' type ='String'/>
</interface.operation>
<interface.operation name = 'traffiCast' ret = ''>
</interface.operation>
</interface>
<interface name.id = 'IMapIVD' type = '' port = 'r' visibility = 'private'>
<interface.description>IVD_Console requires this interface</interface.description>
<interface.operation name = 'calculateRoute' ret = ''>
<operation.param name = 'dest' type ='String'/>
</interface.operation>
<interface.operation name = 'drawRoute' ret = ''>
<operation.param name = 'coord' type ='String'/>
</interface.operation>
<interface.operation name = 'traffiCast' ret = ''>
</interface.operation>
</interface>
<interface name.id = 'IRouteObs' type = '' port = 'p' visibility = 'private'>
<interface.description>Map provides this interface to IVD_Console</interface.description>
<interface.service name = 'ManageObstruction'>
<service.operation name.idref = 'routeObstruction'/>
</interface.service>
<interface.operation name = 'routeObstruction' ret = ''>
<operation.param name = 'route' type ='vector'/>
</interface.operation>
</interface>
<interface name.id = 'IRouteObs' type = '' port = 'r' visibility = 'private'>
<interface.description>IVD_Console requires this interface</interface.description>
<interface.operation name = 'routeObstruction' ret = ''>
```

```
<operation.param name = 'route' type ='vector'/>
</interface.operation>
</interface>
<interface name.id = 'IDataMap' type = '' port = 'p' visibility = 'private'>
<interface.description>Map provides this interface to IVD_Console</interface.description>
<interface.service name = 'ManageEntities'>
<service.operation name.idref ='getEntities'/>
<service.operation name.idref ='getCoord'/>
</interface.service>
<interface.operation name = 'getEntities' ret = ''>
</interface.operation>
<interface.operation name = 'getCoord' ret = ''>
</interface.operation>
</interface>
<interface name.id = 'IDataMap' type = '' port = 'r' visibility = 'private'>
<interface.description>IVD_Console requires this interface</interface.description>
<interface.operation name = 'getEntities' ret = ''>
</interface.operation>
<interface.operation name = 'getCoord' ret = ''>
</interface.operation>
</interface>
<interface name.id = 'IControlCentre' type = '' port = 'p' visibility = 'public'>
<interface.description>CC_Console provides this interface</interface.description>
<interface.service name = 'Login'>
<service.operation name.idref ='login'/>
</interface.service>
<interface.service name = 'ManageDriver'>
<service.operation name.idref ='updateDriver'/>
</interface.service>
<interface.service name = 'ManageMap'>
<service.operation name.idref ='updateMap'/>
</interface.service>
<interface.service name = 'ViewVehicleStatus'>
<service.operation name.idref ='viewStatus'/>
</interface.service>
<interface.operation name = 'login' ret = ''>
<operation.param name = 'username' type ='String'/>
<operation.param name = 'pwd' type ='String'/>
</interface.operation>
<interface.operation name = 'updateDriver' ret = ''>
<operation.param name = 'driverID' type ='String'/>
<operation.param name = 'vehicleID' type ='String'/>
</interface.operation>
<interface.operation name = 'updateMap' ret = ''>
<operation.param name = 'mapID' type ='String'/>
<operation.param name = 'mapEntity' type = 'vector'/>
</interface.operation>
<interface.operation name = 'viewStatus' ret = ''>
<operation.param name = 'coord' type ='String'/>
</interface.operation>
```

```
</interface>
<interface name.id = 'IVehicle' type = '' port = 'p' visibility = 'public'>
<interface.description>IVD_Console provides this interface</interface.description>
<interface.service name = 'SetupConn'>
<service.operation name.idref = 'connect' />
</interface.service>
<interface.service name = 'LoginIVD'>
<service.operation name.idref = 'loginIVD' />
</interface.service>
<interface.service name = 'SearchParking'>
<service.operation name.idref = 'searchParking' />
</interface.service>
<interface.service name = 'Exit'>
<service.operation name.idref = 'exit' />
</interface.service>
<interface.service name = 'WrongTurning'>
<service.operation name.idref = 'reroute' />
</interface.service>
<interface.operation name = 'connect' ret = 'vector'>
<operation.param name = 'mapName' type ='String'/>
</interface.operation>
<interface.operation name = 'loginIVD' ret = ''>
<operation.param name = 'vehicleNo' type ='String'/>
<operation.param name = 'rfidNo' type ='String'/>
</interface.operation>
<interface.operation name = 'searchParking' ret = 'integer'>
<operation.param name = 'dest' type ='String'/>
</interface.operation>
<interface.operation name = 'exit' ret = 'boolean'>
<operation.param name = 'parkID' type ='integer'/>
<operation.param name = 'exitGate' type ='integer'/>
</interface.operation>
<interface.operation name = 'reroute' ret = ''>
<operation.param name = 'coord' type ='String'/>
</interface.operation>
</interface>
<connector name.id = 'IDataCentre -> CC_Console' type = '' role = ''>
<connector.required>
<required.component name.idref = 'CC_Console'/>
<required.interface name.idref = 'IDataCentre'/>
</connector.required>
<connector.provided>
<provided.component name.idref = 'avpsDB'/>
<provided.interface name.idref = 'IDataCentre'/>
</connector.provided>
</connector>
<connector name.id = 'IDataIVD -> IVD_Console' type = '' role = ''>
<connector.required>
<required.component name.idref = 'IVD_Console'/>
<required.interface name.idref = 'IDataIVD'/>
```

```
</connector.required>
<connector.provided>
<provided.component name.idref = 'avpsDB'/>
<provided.interface name.idref = 'IDataIVD'/>
</connector.provided>
</connector>
<connector name.id = 'IMapCC -> CC_Console' type = '' role = ''>
<connector.required>
<required.component name.idref = 'CC_Console'/>
<required.interface name.idref = 'IMapCC'/>
</connector.required>
<connector.provided>
<provided.component name.idref = 'Map'/>
<provided.interface name.idref = 'IMapCC'/>
</connector.provided> </connector>
<connector name.id = 'IMapIVD -> IVD_Console' type = '' role = ''>
<connector.required>
<required.component name.idref = 'IVD_Console'/>
<required.interface name.idref = 'IMapIVD'/>
</connector.required>
<connector.provided>
<provided.component name.idref = 'Map'/>
<provided.interface name.idref = 'IMapIVD'/>
</connector.provided>
</connector>
<connector name.id = 'IDataMap -> Map' type = '' role = ''>
<connector.required>
<required.component name.idref = 'Map'/>
<required.interface name.idref = 'IDataMap'/>
</connector.required>
<connector.provided>
<provided.component name.idref = 'avpsDB'/>
<provided.interface name.idref = 'IDataMap'/>
</connector.provided>
</connector>
<connector name.id = 'IRouteObs -> IVD_Console' type = '' role = ''>
<connector.required>
<required.component name.idref = 'IVD_Console'/>
<required.interface name.idref = 'IRouteObs'/>
</connector.required>
<connector.provided>
<provided.component name.idref = 'CC_Console'/>
<provided.interface name.idref = 'IRouteObs'/>
</connector.provided></connector>
</iXML>
```

E3. SMART

Quality concern weights and design template contributions calculated using SMART in Table E3.1 for Scenario 1 of GVPS.

Table E3.1 SMART for GVPS - Scenario 1

Concern	Sub-Concern	Scope	S1: *ClusterServer* Pattern				S2: *Proxy* Pattern			
			s.	s*		μ	s.	s*		μ
Efficiency	Memory	S2.6.2	2	0.09	0.17	0.521	3	0.13	0.26	0.824
	Processor	S2.6.2	2	0.09			3	0.13		
Flexibility	Expendability	S1.2.1	N/A	0.00	0.00		3	1.00	0.17	
Performance	ResponseTime_UPL	S1.2.1	2	0.67	0.24		2	0.67	0.24	
	ResponseTime_PL	S1.2.1	2	0.67			2	0.67		
Reliability	Availability	S1.5.1	3	1.00	0.12		0	0.00	0.00	
Security	Integrity	S1.2.1	1	0.06	0.12		3	1.00	0.35	
	Integrity	S2.1.1	1	0.06			3	1.00		

Appendix F:

Design Templates

Design templates of *ClusteServer* pattern, *Proxy* pattern, *ServiceOrderProvision* local scheme, *Three-tier proxy server* architectural style are shown in Table F1.1 – F1.4

Table F1.1 *ClusterServer* template

Category	Pattern
Name	*ClusterServer*
Also-Known-As	-
Related-Rules	-
Intent	This patterns cluster starts off with Server Clustering, which focuses on using server clusters to design an infrastructure tier that meets specific availability and scalability requirements. A server cluster is two or more servers that are interconnected to form a unified virtual computing resource.
Context	Clustering servers increases the availability of a system by ensuring that if a server becomes unavailable because of failure or planned downtime, another server in the cluster can assume the workload, ensuring that the application remains available to users. Clustering also enhances scalability by supporting more users at the current level of performance or by improving application performance for the current users.
Motivation	An enterprise application has to meet ever-increasing operational demands, including higher availability, improved performance, and the ability to maintain these demands as the load on applications increases. This creates the need for application and supporting infrastructure designs that maximize scalability and availability.

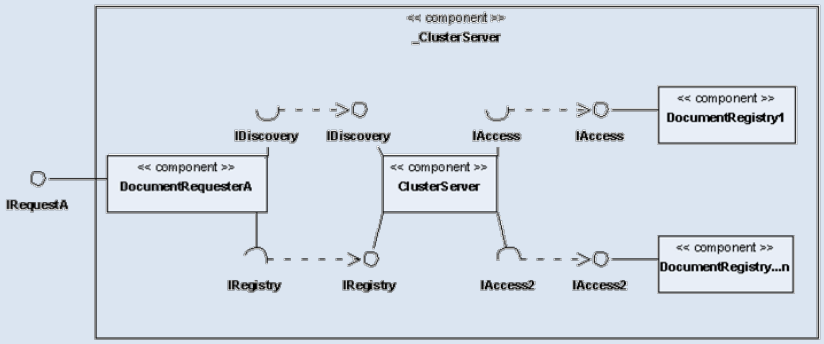
Configuration	
Consequences	Performance.Response time = {*ClusterServer* is maintaining performance., M} Performance.Throughput = { *ClusterServer* is maintaining performance., M} Reliability.Availability = {*ClusterServer* is improving availability using active redundancy and automatic restart during failover., H} Maintainability.Requirement = {*ClusterServer* complexity may compromise system maintainability., L} Maintainability.Technology = {*ClusterServer* complexity may compromise system maintainability., L} Maintainability.Time = {*ClusterServer* complexity may compromise system maintainability., L} Security.Integrity = {Clustering onto two or more server may comprise integrity of data., L} Efficiency.Memory = {ClusterServer provides space optimisation through resources sharing, M} Efficiency.Proccesor = {ClusterServer provides time optimisation through resources sharing, M}

Table F1.2 *Proxy* pattern template

Category	Pattern
Name	Proxy
Also-Known-As	Surrogate
Related-Rules	Decorator, Adapter
Intent	The pattern makes the clients of a component communicate with a representative rather than to the component itself. Introducing such a placeholder can serve many purposes, including enhanced efficiency, easier access and protection from unauthorised access.
Context	Proxy is applicable whenever there is a need for more versatile or sophisticated reference a component. Some common situations in which the pattern is applicable: 1. Remote proxy – where clients of remote components should be shielded from network addresses and inter-process communication protocols. 2. Protection proxy – where components must be protected from unauthorised access 3. Cache proxy – where multiple simultaneous access to a component must be synchronised 4. Counting proxy – where accidental deletion of components must be prevented or usage statistic collected 5. Virtual proxy – where the processing or loading of a component might costly, while partial information about the component might be sufficient

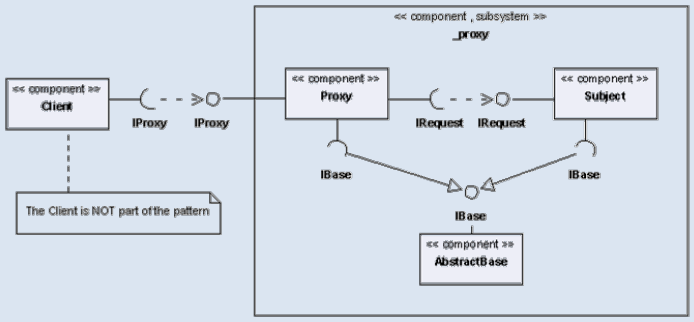
	6. Firewall Proxy – where local clients should be protected from the outside world
Motivation	One reason for controlling access to a component is to defer the full cost of its usage until we actually need it. Until that point we can use some light objects (proxies) exposing an identical interface as the heavy objects to the Client. When the proxy is accessed it forwards the request to the real subject. This ability to control the access to a component can be required for a variety of reasons: caching, access control, synchronisation, lazy creation, remote access.
Configuration	
Consequences	Efficiency.Memory = {The proxy provides space optimisation through caching and lazy construction when the cost of data access and rendering is reduce, H} Efficiency.Proccesor = {The proxy provides time optimisation through caching and lazy construction when the cost of data access and rendering is reduce, H} Performance.ResponseTime = {A virtual proxy helps to implements a 'load-on-demand strategy' that avoid unnecessary loads and usually speeds up the application, however complex implementation would cause less efficiency due to indirection, M} Reusability.Modularity = {The proxy provides weak coupling between clients and subsystems, M} Flexibility.Expendability = {A remote proxy decoupling clients from the locations of remote server components, H} Security.Integrity = {Protection proxy and smart references allow additional housekeeping tasks when a component is accessed, H}

Table F1.3 *ServiceOrder Provision* template

Category	Local scheme
Name	Service-Order Provision
Also-Known-As	Order Provision
Related-Rules	-
Intent	A document may require a search, locate and order service. This design template ensures that the order service resides in a component that is separate from search and locate services.
Context	When the document manager requires search, locate and order services, restricting document order in a separate execution is good a strategy. A requestor component can obtain document and location identifiers from a centralized document registry before placing a document order. Document orders are placed with the document supplier component.
Motivation	DocumentManager may require services of DocumentServer which consists of ISearch and ILocate, and DocumentServer which consists of IOrder.

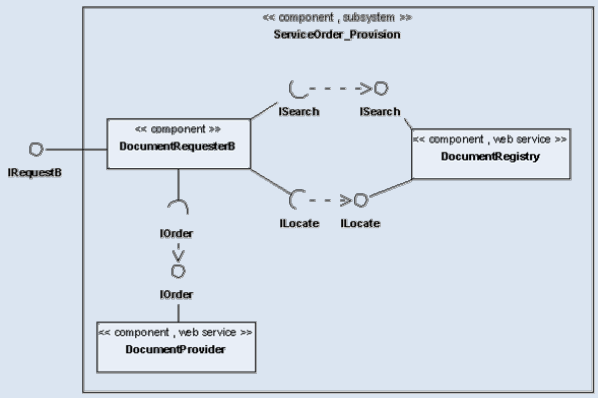
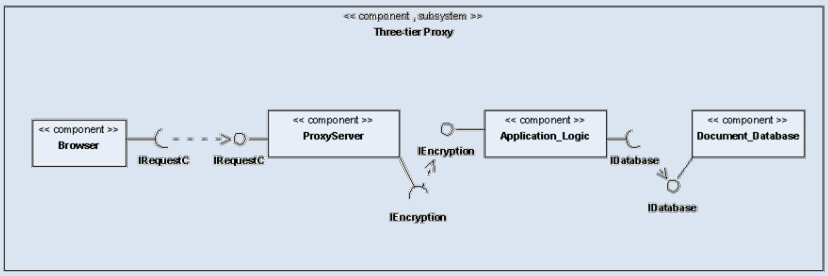
Configuration	
Consequences	Performance.ResponseTime = {contributes flexibility in the communication with document provider , M} Performance.Throughput = {contributes flexibility in the communication with document provider, M} Maintainability.Time = {contributes towards maintenance time for the document main services, H} Maintainability.Requirement = {allows the document server maintain the order service more effectively, H} Reliability.Availability = {improves the availability of related services which allows longer duration of order service to be served, H}

Table F1.4 *Three-tier proxy server* template

Category	Style
Name	*Three-tier proxy server*
Also-Known-As	Three-Tier Client/Server Architecture
Related-Rules	-
Intent	A tier is a partitioning of functionality that may be allocated to a separate physical machine (i.e. web browsers client, web server and database server) which improves maintainability while hiding the complexity of distributed processing.
Context	When we have to design applications for distributed enterprise information systems where usually some desktop components will access or modify shared resources, mostly located within a non-active database.
Motivation	Partition application functionality into three tiers: front-end clients, application servers (domain server) and a database storage. The front-end clients tier consists of cosmponents unique to every user include application specific logic & the user interface. The application server tier, supported by a multi-user environment, holds the shared parts of application & bussiness logic. This tier needs services like transaction, concurrency control & security. The task of the database storage tier is to manage persistency of certain data/info and to execute the database transaction.

Configuration	
Consequences	Performance.ResponseTime = {The requests from individual browsers may first arrive at a proxy server, which exists to improve the performance of the Web-based system, M} Performance.Throughput = {The proxy server is typically located close to the users, often on the same network, so that they save tremendous amount of both communication and computation resources, M} Reliability.Availability = {Functionality routine accessibility, M} Maintainability.Requirement = {A tier is a partitioning of functionality improves maintainability while hiding the complexity of distributed processing, H} Maintainability.Technology = {A tier is a partitioning of functionality improves maintainability of technology require, H} Maintainability.Time = {The partitioning of functionality allows components loosely coupling hence improve time to maintains it, H} Security.Integrity = {The proxy server is also used to restrict users's access to certain Web sites. In this case the proxy server is acting somewhat like a firewall, H}

Appendix G:

Quality Descriptions

Quality descriptions are shown in Table G.1.

Table G.1 Quality descriptions

Concern	SubConcern	Unit Type*	Unit Name	Notes
Performance	ResponseTime_upl	N	Seconds	Performance of response time under-peak load.
Performance	ResponseTime_pl	N	Seconds	Performance of response time during peak load.
Performance	Throughput_upl	N	Trans/per second	Performance of throughput under-peak load.
Performance	Throughput_pl	N	Trans/per second	Performance of throughput during peak load.
Reliability	Availability	N	%	Reliability of availability according to service access time.
Maintainability	Requirement	V		Maintainability of requirement refers to the role of stakeholder who is able to request for maintaining architectural components.
Maintainability	Time	N	Months	Maintainability of time refers to elapse time for maintaining the architectural components.
Maintainability	Technology	V		Maintainability of technology refers to technology require for maintaining architectural components.
Component	Standard	V		Component standard protocol
Component	Cost	N	GBP	Component cost charge yearly

Concern	SubConcern	Unit Type*	Unit Name	Notes
Component	Version	N		Component version
Component	Availability	V		Component availability
Component	Certification	B		Component certification
Component	In	N	Required	Component required interfaces
Component	Out	N	Provided	Component provided interfaces
Component	Services	V		Component tagged services
Business	Cost	V		Business cost intensity
Business	Schedule	N	Months	Business schedule intensity
Business	Platform	V		Business platform
Security	Integrity	V		Security of integrity refers to the extent to which access to software or data by unauthorised persons can be controlled.
Flexibility	Expendability	V		Flexibility of expendability refers to the degree and effort to which the program can be extended.
Reusability	Modular	V		Reusability of modular refers to the functional independence of program components.
Efficiency	Memory	N	%	Efficiency of memory refers to the scarce resource is effectively uses.
Efficiency	Processor	N	%	Efficiency of memory refers to the scarce resource is effectively uses.

Legends: N – Numeric V – Verbal B – Boolean

Quality Definitions

Quality definitions described below are adopted from [Iso01][McCall77]:

- *Efficiency* – Efficiency is refers to the level of use of scarce computational resources such CPU cycles and memory.
  - Memory: Memory involves space and time spent using the resources.
  - Processor: Processor involves space and time spent using the resources.
- *Performance* – Performance is about timing, events occur and the system must respond to them.
  - Response Time: Managing the interprocess communication volume and data access frequencies

- Throughput: The speed with which a component processes data.

- *Reliability* – Reliability is concerned with system failure and its associated consequences. A system failure occurs when the system no longer deliver consistent with its specification.
  - Availability: Availability is concerned with the proportion of elapsed time that the component is able to be used.
- *Maintainability* – Maintainability refers to the change which can occur to any aspect of a system.
  - Requirement: Maintainability of requirement refers to the role of stakeholder who is able to request for maintaining architectural components.
  - Time: Maintainability of requirement refers to elapse time for maintaining the architectural components.
  - Technology: Maintainability of requirement refers to technology require for maintaining architectural components.
- *Flexibility* – Flexibility refers to the effort required to modify an operational program (or part thereof).
  - Expendability – Flexibility of expendability refers to the degree and effort to which the program can be extended.
- *Reusability* – Reusability is the ease with which an existing component can be reused
  - Modularity – Reusability of modularity refers to the functional independence of program components.
- *Security* – The ability to prevent unauthorized access to program or data
  - Integrity – Security of integrity refers to the extent to which access to software or data by unauthorised persons can be controlled.

Glossary

ADL	Architecture Description Language
AHP	Analytic Hierarchical Process
ASAAM	Aspectual Software Architecture Analysis Method
ATAM	Architecture Trade-off Analysis Method
CADL	Component Architecture Description Language
CBD	Component-based System Development
CBSE	Component-based Software Engineering
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
CSAFE	Component-based Software Architecture analysis FramEwork
DFC	Distributed Feature Composition
EDDIS	Electronic Document Delivery Interchange System
EJB	Enterprise JavaBeans
FOSS	Free cOmponentS Open Source

GC	Garbage Collector
GUI	Graphical User Interface
GVPS	Guided Vehicle Parking System
IVD	In-Vehicle-Device
JDK	Java Development Kit
JVM	Java Visual Machine
NEC	National Electronic Company
NFR	Non-Functional Requirement
OCS	Open Control System
OTS	Off-The-Shelf
PID	Process ID
POS	Point-Of-Sales
SMART	Simple Multi-Attributes Rating Technique
XML	Extensible Markup Language

References

[Abowd97] Abowd, G., Bass, L., Clements, P., Kazman, R. and Northrop, L.: *Recommended Best Industrial Practice for Software Architectural Evaluation*. Technical Report CMU/SEI-96-TR-025. Pittsburgh, PA, Software Engineering Institute. Carnegie Mellon University. 1997.

[Admodisastro06] Admodisastro, N. and Kotonya, G.: *Towards an Integrated Approach to Architectural Analysis in Component-based Software Development.* Proceeding of the Work in Progress Session in the 32nd IEEE EuroMicro Conference. 2006.

[Admodisastro08] Admodisastro, N. and Kotonya, G.: *Architectural Analysis Approaches: A Component-Based System Development Perspective*. Proceeding of the International Conference on Software Reuse (ICSR). Springer-Verlag, Berlin Heidelberg, 2008; LNCS 5030: 26-38.

[Admodisastro10] Admodisastro, N. and Kotonya, G.: *An Architectural Analysis Approach for Black-box Component-Based Systems*. Proceeding of the 2nd GSTF International Conference on Software Engineering (SE), Phuket, Thailand. 2010; 68-74.

[Admodisastro11a] Admodisastro, N., Kotonya, G.: *An Architecture Analysis Approach for Supporting Black-box Software Development*. Proceeding of the

European Conference on Software Architecture. Springer-Verlag, Heidelberg, 2011; LNCS 6903: 180-189.

[Admodisastro11b] Admodisastro, N. and Kotonya, G.: *Usability Requirements for Architectural Analysis Tool to Support CBD*. Proceeding of the 2nd International Conf. User Science and Engineering (i-USer). IEEE Computer Society, 2011; 118-123.

[Advant10] Advant: *ABB Control Systems.* Available: http://www.abb.com/controlsystems, 2010.

[Aoyama 01] Aoyama, M.: CBSE in Japan and Asia. In G. T. Heineman and W.T. Council, *Component-Based Software Engineering: Putting the Pieces Together*. Addison-Wesley. 2001.

[Aoyama98] Aoyama, M.: *New Age of Software Development: How Component-based Software Engineering Changes the Way of Software Development*. International Workshop on Component-Based Software Engineering, 1998.

[Auto10] Auto: Automotive-Articles. Available: http://www.innerauto.com/Automotive_Articles/, 2010.

[Babar04a] Babar, M.A. and Gorton, I.: *Comparison of Scenario-Based Software Architecture Evaluation Methods*. Proceeding of the Asia-Pacific Software Engineering (APSEC). IEEE Computer Society, Washington D.C., 2004; 600-607. DOI: 10.1109/APSEC.2004.38.

[Babar04b] Babar, M. A., Zhu, L. and Jeffery, R.: *A Framework for Classifying and Comparing Software Architecture Evaluation Methods*. Proceeding of the 2004 Australian Software Engineering Conference (ASWEC). IEEE Computer Society, 2004; 309-318. DOI: 10.1109/ASWEC.2004.1290484.

[Babar07] Babar, M. A. and Gordon, I.: *A Tool for Managing Software Architecture Knowledge*. Proceeding of the 2nd Workshop on SHAring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent (SHARK). IEEE Computer Society, 2007; 11-17. DOI: 10.1109/SHARK-ADI.2007.1

[Barbacci05] Barbacci, M.R.: *SEI Architecture Analysis Techniques and When to Use Them*. CMU/SEI-2001-TN-005. Carnegie Mellon Software Engineering Institute. 2005.

[Bashroush04] Bashroush, R., Spence, I., Kilpatrick, P. and Brown, T. J.: Towards and Automated Evaluation Process for Software Architectures. *IASTED on Software Engineering*, 2004; 418: 182.

[Bass05] Bass, L., Clements, P. and Kazman, R.: *Software Architecture in Practice*. 2nd Ed. SEI Series in Software Engineering. Addison Wesley. 2005.

[Becker06] Becker, S., Brogi, A., Gorton, I., Overhage, S. and Romanovsky, A. and Tivoli, M.: *Towards an Engineering Approach to Component Adaptation*. R. H. Reussner et al. (Eds.): Architecting Systems. Springer-Verlag, Berlin Heidelberg 2006; LNCS 3938: 193-215. DOI: 10.1007/11786160.

[Bond05] Bond, G.W., Cheung, E., Goguen, H.H., Hanson, K.J., Henderson, D., Karam, G.M., Purdy, K.H., Smith, T.M., Zave, P.: *Experience with Component-Based Development of a Telecommunication Service*. Proceeding of the ACM Sigsoft Symposium on Component-Based Software Engineering (CBSE). Springer-Verlag, Berlin Heidelberg, 2005; LNCS 3489: 298-305.

[Britannica10] Encyclopædia Britannica: Science & Technology: Engineering. Available: http://www.britannica.com/EBchecked/topic/187549/engineering, 2010.

[Brown96] Brown, A. and Wallnau, K.: *Engineering of Component-based System*. Proceedings of the 2nd IEEE International Conference on Engineering of Complex Computer Systems (ICECCS), 1996; 7-15.

[Buschmann96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture: System of Patterns*. Vol. 1. Wiley Series in Software Design Pattern, Wiley. 1996.

[Chung95a] Chung, L., Nixon, B. and Yu, E.: *An Approach to Building Quality into Software Architecture*. Proceeding of the Conference of the

Centre for Advanced Studies on Collaborative Research (CASCON). IBM Press, Canada, 1995; 13-25.

[Chung95b] Chung, L. and Nixon, B. A.: *Dealing with Non-Functional Requirements: Three Experimental Studies of Process-Oriented Approach*. Proceeding of International Conference on Software Engineering (ICSE). ACM Press 1995; 25-37. DOI: 10.1145/225014.225017.

[Clements95] Clements, P.: From Subroutines to Subsystems: Component-Based Software Development. *The American Programmer*, 1995; 8(11).

[Clements96] Clements, P., and Northrop, L.: *Software Architecture: An Executive Overview*. CMU Technical report CMU/SEI-96-TR-003. Pittsburgh, PA, Software Engineering Institute. Carnegie Mellon University. 1996.

[Coplien97] Coplien, J.O.: Idioms and Patterns as Architectural Literature. *IEEE Software: Special Issue on Objects, Patterns and Architectures*, 1997; 14(1): 36-42.

[Crnkovic02] Crnkovic, I., Larsson, M. (Editors): *Building Reliable Component-Based Software Systems*. Artech House Publisher. 2002.

[Cs10] CS240: *Group Project Software Design/Project Skills*. Available: http://www.comp.lancs.ac.uk/~andreas/Teaching_AUM.htm , 2010 [10/21/2010].

[Darwin95] Darwin J. M., Dulay, N., Eisenbach, S., and Kramer, J.: *Specifying Distributed Software Architectures.* Proceeding of the 5th European Software Engineering Conference (ECSA). 1995; 137-153.

[Dashofy02] Dashofy, E.M.,Hoek, A.v.d., and Taylor, R.N. *An Infrastructure for the Rapid Development of XML-based Architecture Description Languages*. Proceedings of the 24th International Conference on Software Engineering (ICSE). 2002; 266-276.

[Dobrica02] Dobrica, L. and Eila, N.: A Survey on Software Architecture Analysis Methods. *IEEE Transaction on Software. Engineering*. 2002; 28(7): 638-653. DOI: 10.1109/TSE.2002.1019479.

[Ekstedt02] Ekstedt, M. and Johnson, P.: *Exploring Architectural Analysis Credibility from a Developer Perspective*. Proceeding on the

Australasian Workshop on Software and System Architecture (AWSA). 2002. DOI: 10.1.1.16.1268

[Fayad97] Fayad, M. and Schmidt, D. C.: Object-Oriented Application Frameworks. Communications of the ACM, 1997; 40(10): 32-38.

[Feblowitz98] Feblowitz, M. D. and Greenspan, S. J.: Scenario-Based Analysis of COTS Acquisition Impacts. *Requirements Engineering*, 1998; 3(3-4): 182-201.

[Feller02] Feller, J. and Fitzgerald, B.: *Understanding Open Source Software Development*. Addison-Wesley. 2002.

[Gamma95] Gamma, E., Helm, R., Johnson, R. and Vlissides, J.: *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional. 1995.

[Garlan97] Garlan, D., Monroe, R., and Wile, D.: *ACME: An Architecture Description Interchange Language*. Proceedings of the 1997 Conference of the Centre for Advanced Studies on Collaborative Research (CASCON), 1997; 169-183.

[Gillies96] Gillies, A.: *Software Quality: Theory and Management*. International Thomson Computer Press. 1996.

[Grau05] Grau, G., Franch, X., Maiden, N. A. M.: *REDEPEND-REACT: an Architecture Analysis Tool*. Proceeding IEEE International Conference on Requirement Engineering (RE). IEEE Computer Society 2005; 455-456. DOI: 10.1109/RE.2005.55.

[Heineman01] Heineman, G. T. and Council, W.T.: *Component-Based Software Engineering: Putting the Pieces Together*. Addison-Wesley. 2001.

[Hutchinson05] Hutchinson, J., and Kotonya, G.: *Patterns and Component-Oriented System Development*. Proceeding of the EuroMicro Conf. on SEAA, 2005; 126-133.

[Hutchinson06] Hutchinson, J., and Kotonya, G.: *A Review of Negotiation Techniques in Component-Based Software Engineering*. Proceeding of the EuroMicro Conf. on SEAA. IEEE Computer Society, Washington D.C. 2006; 152-159. DOI: 10.1109/EUROMICRO.2006.12

[Iso01] International Standard: *ISO/IEC 9126-1*. Institute of Electrical and Electronic Engineers, P*art 1,2,3: Quality Model*. Available: http://www.iso.ch [2001]

[Jacobson97] Jacobson, I., Griss, M. and Jonsson, P.: *Software Reuse: Architecture Process and Organization for Business Success*. Addison-Wesley, Reading. 1997.

[Java10] Java SE Desktop Technologies. Introducing Java Beans. Available: http://java.sun.com/developer/onlineTraining/Beans/Beans1/ [2010]

[Java11] Java SE Documentation, Java VisualVM. Available: http://download.oracle.com/javase/6/docs/technotes/guides/visualvm/index.html [2011]

[Kazman96] Kazman, R.: Tool *Support for Architecture Analysis and Design*. Joint Proceedings of the 2nd International Software Architecture Workshop (ISAW-2) & International Workshop on Multiple Perspectives in Software Development on SIGSOFT 1996 Workshops. ACM Press. New York, USA, 1996; 94-97. DOI: 10.1145/243327.243618

[Kazman98] Kazman, R., Klein, M., Barbacci, M., Longstaff, T., Lipson, H. and Carriere, J.: *The Architectural Tradeoff Analysis Method*. Proceeding of IEEE International Conference on Engineering of Complex Computation System (ICECCS), 1998; 68-78. DOI: 10.1109/ICECCS.1998.706657.

[Khosravi04] K. Khosravi, and Y.G. Guéhéneuc. *Quality Model for Design Patterns*. Summer 2004.

[Klein99] Klein, M. and Kazman, R.: *Attribute-Based Architectural Styles*. Technical Report CMU/SEI-99-TR-22. Pittsburgh, PA, Software Engineering Institute. Carnegie Mellon University. 1999.

[Kotony03] Kotonyo, G., Sommerville, I. and Hall, S.: *Towards a Classification for Component-Based Software Engineering Research*. Proceeding of the 29th IEEE EuroMicro Conference. 2003; 43-52.

[Kotonya04a] Kotonya, G. and Hutchinson, J.: *Viewpoints for Specifying Component-Based Systems*. Component-Based Software Engineering, Proceeding of the 7th International Symposium on Component-Based

Software Engineering (CBSE). Springer-Verlag, Berlin, 2004; LNCS 3054: 114-121.

[Kotonya04b] Kotonya, G., Hutchinson, J. and Bloin, B.: COMPOSE: Method for Formulating and Architecting Component and Service–Oriented Systems. In Z. Stojanovic, and A. Dahanayake, eds., *Service-Oriented Software System Engineering: Challenges and Practices*. Idea Group Inc. 2004.

[Kotonya05a] Kotonya, G, Hutchinson, J. *Managing Change in COTS-Based Systems*. Proceeding of the IEEE International Conference on Software Maintenance (ICSM). IEEE Computer Society, Washington D.C., 2005; 69-78. DOI: 10.1109/ICSM.2005.61.

[Kotonya05b] Kotonya, G. and Hutchinson, J.: *Analysing the Impact of Change in COTS-Based Systems*. Proceeding of the ICCBSS. Springer-Verlag, Heidelberg, 2005; LNCS: 3412: 212-222. DOI: 10.1007/b105900

[Kotonya07] Kotonya, G, Hutchinson, J. A.: *Service-Oriented Approach for Specifying Component-Based Systems*. Proceeding of the ICCBSS. Springer-Verlag, Heidelberg, 2007; LNCS 3412: 150-162. DOI: 10.1109/ICCBSS.2007.4

[Kotonya08] Kotonya, G.: *An Architecture-Centric Development Environment for Black-Box Component-Based Systems*. Proceeding of the European Conference on Software Architecture (ECSA). Springer-Verlag, Heidelberg, 2008; LNCS 5292: 98-113.

[Kung-Kiu04] Kung-Kiu, L: *Component-Based Development Case Studies*. World Scientific, 2004.

[Kurpjuweit02] Kurpjuweit, S.: A Family Tools to Integrate Software Architecture Analysis and Design. PhD Thesis. Software Engineering Institute. Carnegie Mellon University, USA. 2002.

[Lau07] Lau, L. and Wang, Z.: Software Component Model. *IEEE Transaction on Software Engineering*; 2007; 33(10): 709-124.

[Li08] Li, J. et al.: A State-of-the-Practice Survey of Risk Management in Development with Off-the-Shelf Software Components. *IEEE Transactions on Software Engineering*, 2008; 34(2).

[Luckham95] Luckham, D. C. et al.: Specification and Analysis of System Architecture Using Rapide. *IEEE Transactions on Software Engineering*, 1995; 21(6): 336-354. DOI: 10.1109/32.385971.

[Lüders00] Lüders, F.: Architectural Styles in Component-Based Software Engineering. Seminar in Component-Based Software Engineering: State of the Art. Mälarden University. Västerås, Sweden. 2000.

[Luer01] Luer, C., and Rosenblum S. D.: WREN an Environment for Component-Based Development. *ACM SIGSOFT Software Engineering Notes*, 2001; 26(5): 207-217.

[McCall77] J.A. McCall, P.K. Richards, and G.F. Walters: *Factors in Software Quality*. RADC-TR-77-369. US Department of Commerce, 1977.

[Medvidovic00] Medvidovic, N. and Taylor, R. N.: A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Transaction Software Engineering*, 2000; 26(1): 70-93.

[Medvidovic02] Medvidovic, N. et al.: Modeling Software Architectures in the Unified Modeling Language. *ACM Transactions on Software Engineering and Methodology,* 2002; 11(1): 2-57.

[Medvidovic07] Medvidovic, N. and Dashofy, E.M.: Moving Architectural Description from Under the Technology Lamppost. *Information and Software Technology,* 2007; 49(1): 12-31. DOI: 10.1016/j.infsof.2006.08.006.

[Medvidovic96] Medvidovic, N., Oreizy, P., Robbins, J. E. and Taylor, R. N.: *Using Object-Oriented Typing To Support Architectural Design in the C2 Style*. Proceeding of ACM SIGSOFT'96: 4th Symposium on the Foundations of Software Engineering. ACM Press, New York 1996; 24-32. DOI: 10.1145/250707.239106.

[Msdn10] MSDN: Performance and Reliability Pattern. Available: http://msdn.microsoft.com/en-us/library/ff648802.aspx, 2010 [Jan. 01, 2011].

[Obbink07] Obbink, H, Kruchten, P, Kozaczynski, W, Hilliard, R, Ran, A., Postema, H., Lutz, D., Kazman, R., Tracz, W., Kahane, E.: *Software Architecture Review and Assessment (SARA) Report*. Available:

http://kruchten.com/philippe/architecture/SARAv1.pdf [6 December 2007]

[Papazoglou08] Papazoglou, M.P.: *Web Services: Principles and Technology*. Prentice-Hall. 2008.

[Perry92] Perry, D.E., and Wolf, A. L.: Foundations for the Study of Software Architecture. *ACM SIGSOFT Software Engineering Notes*, 1992; 17(4): 40-52.

[Persse01] Persse, J. R.: *Implementing the Capability Maturity Model*. Wiley, New York. 2001.

[Pilone05] Pilone, D., Pitman, N.: *UML 2.0 in a Nutshell*. Oreilly. 2005.

[Pressman09] Pressman, R.: *Software Engineering: A Practitioner's Approach*. 7th Ed. McGraw Hill. 2009.

[Rami03] Rami, B., and Wolfgang, E.: *Evaluating Software Architectures: Development, Stability and Evolution*. Proceeding of ACS/IEEE International Conference on Computer System and Applications (AICCSA). IEEE Computer Society Press 2003; 47-56. DOI: 10.1109/AICCSA.2003.1227480.

[RedependReact07] REDEPENDREACT: The REDEPEND-REACT Homepage. Available: http://www.lsi.upc.es/~ggrau/REDEPEND-REACT/index.html [2 December 2007].

[Saaty90] Saaty, T.L.: *The Analytic Hierarchy Process*. McGraw-Hill, New York. 1990.

[Saniabille01] Saniabille, R., Favre, J-M., and Ledru, Y.: *Helping Various Stakeholders to Understand a Very Large Component-Based Software*. Proceeding of the 27th IEEE EuroMicro, 2001; 104-111.

[Shaw96] Shaw, M. and Garlan, D.: *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall. 1996.

[Shepetukha01] Shepetukha, Y., and Olson, D. L.: Comparative Analysis of Multiattribute Techniques Based On Cardinal and Ordinal Inputs. *Mathematical Computing Modelling*, 2001: 34: 229-241.

[Sommerville10] Sommerville, I.: *Software Engineering*. 9th Ed. Addison-Wesley. 2010.

[Spagnoli06] Spagnoli, L., Almeida, I., Becker, K., Blois, A. P., and Werner, C.: *Adaptation and Composition within Component Architecture Specification*. Proceeding of the International Conference on Software Reuse (ICSR). Springer-Verlag, Berlin Heidelberg, 2006; LNCS 3140: 142-155. DOI: 10.1007/11763864.

[Stafford01a] Stafford, J., and Wolf, A.: Software Architecture. pp. 371-388. In G. T. Heineman and W.T. Council, *Component-Based Software Engineering: Putting the Pieces Together*. Addison-Wesley. 2001

[Stafford01b] Stafford, J.A., Richardson, D.J. and Wolf, A. L.: Architecture-Level Dependence Analysis for Software Systems. *International Journal of Software Engineering & Knowledge Engineering*. 2001; 11(4): 431-451. DOI: 10.1.1.40.6873.

[Stafford98] Stafford, J. A., Richardson, D. J. and Wolf, A. L.: *Aladdin: A Tool for Architecture-Level Dependence Analysis of Software Systems*. Technical Report CU-CS-858-98. University of Colorado, 1998.

[Summers06] M. Summers, *240 Report*. Technical Report. Computing Department, Lancaster University. 2006.

[Tekinerdogan04] Tekinerdogan, B.: *ASAAM: Aspectual Software Architecture Analysis Method*. Proceeding on Working IEEE/IFIP Conference on Software Architecture (WICSA). IEEE Computer Science, Washington, D.C., 2004; 5-14. DOI: 10.1109/WICSA.2004.1310685

[Tran99] Tran, Q. and Chung, L.: *NFR-Assistant: Tool Support for Achieving Quality*. Proceeding of IEEE Symposium on Application – Specific System and Software Engineering and Technical (ASSET). IEEE Computer Society, 1999; 284-289. DOI: 10.1109/ASSET.1999.756782

[Uml01] Unified Modeling Language. *UML® Resource Page*. Last updated on, 2010. Available: http://www.uml.org/, 2010 [Oct. 21, 2010]

[Upadhyaya08] Upadhyaya, B.P.: *Component Based Software Development - An Industrial Experience with a Labour Market Information System*. Proceedings of 19th Australian Software Engineering Conference (ASWEC), 2008; 497-506.

[van den Brand01] van den Brand, M.G.J., Heering, J., de Jong, H.A., de Jonge, M., Kuipers, T., Klint, P., Moonen, L., Olivier, P.A., Scheerder, J., Vinju, J.J., Visser, E., Visser, J.: *The ASF+SDF Meta-Environment: a Component-Based Language Development Environment*. Proceedings of 10th International Conference on Computational Complexity (CCC), 2001; 365-370.

[Vieira00] Vieira, M. E. R., Dias, M. S., and Richardson, D. J.: *Analyzing Software Architecture with Argus-I*. Proceeding of the International Conference Software Engineering (ICSE). ACM Press, New York, USA, 2000; 758-761. DOI: 10.1109/ICSE.2000.870489.

[Vigder01] Vigder, M.: The Evolution, Maintenance and Management of Component-Based Systems. pp. 527-539. In G. T. Heineman and W.T. Council, *Component-Based Software Engineering: Putting the Pieces Together*. Addison-Wesley. 2001.

[Vigder96] Vigder, M., Gentleman, M., Dean, J.: *COTS Software Integration: State of the Art. Institute for Information Technology*. National Research Council. 1996.

[Volgyesi02] Volgyesi, P., Ledeczi, A.: *Component-Based Development of Networked Embedded Applications*. Proceedings of 28th IEEE EuroMicro Conference on Component-Based Software Engineering, 2002; 68-73.

[Wallnau02] Wallnau, K.C., Hissam, S.A. and Seacord, R.C.: *Building System from Commercial Components*. SEI Series in Software Engineering. Addison-Wesley, Reading. 2002.

[Wallnau03] Wallnau, K.C.: Volume III: *A Technology for Predicable Assembly from Certifiable Components*. Technical Report CMY/SEI-2003-TR-009. Pittsburgh, PA, Software Engineering Institute. Carnegie Mellon University. 2003.

[Weiss01] Weiss, M.: *Patterns and Non-Functional Requirements*. Technical Paper. Carleton University. 2001.

[Wikipedia10] Wikipedia: Software Component. Available: http://www.wikipedia.org/wiki/Software_component, 2010.

[Xml10] XML and XMI. *CORBA®, XML and XMI®*. Last updated on Available: http://www.omg.org/technology/xml/index.htm, 2010 [June 25, 2009].