Vrije Universiteit Brussel

Faculteit Wetenschappen
Vakgroep Computerwetenschappen

Workflow Patterns for Modeling Computer Aided Engineering Iterations

Proefschrift ingediend met het oog op het behalen
van de graad van Master in de Ingenieurswetenschappen: Computerwetenschappen

Wouter De Geest

Promotor: Prof. Dr. Viviane Jonckers
Begeleider: Niels Joncheere

AUGUSTUS 2009

Abstract

Worfklow languages are typically used for modelling business processes since it is very natural to think of business processess as being workflows. But we should not limit workflow languages to business processes only. Any logical flow of tasks can in theory be modelled by a workflow language. It might be that this flow of tasks, or process, is a complex one and that as a result current workflow approaches might need adaptation or added functionality. One such complex flow is found in Computer Aided Engineering (CAE) and Multi-Disciplinary Optimization (MDO). Applications in these domains differ in complexity from applications in other domains in such a way that they cannot be built using today's workflow languages. This complexity has been studied by SSEL (Software and Systems Engineering Lab, VUB). The first outcome of this study was the recognition of three main requirements: modularization of crosscutting concerns, advanced data support and advanced iteration support. A second outcome was the proposal of a conceptual workflow language meeting the first two requirements. This thesis contributes to the framework support for the third and final requirement of supporting iterations. Iterations are algorithms such as optimization methods, Monte-Carlo methods and many more. In a first step we will define these iterations by making a classification. In a second step we propose workflow patterns that fit into the existing approach and that allow to implement the classified iterations. The result of this thesis makes the conceptual workflow language meet all requirements. This means that a first implementation of the language can start. For the first time MDO applications can be developed using a real workflow language where they gain all direct benefits from workflow languages: evolution and reuse, customizability, collaboration and distributive computation.

Samenvatting

Workflowtalen worden in het bijzonder gebruikt om bedrijfsprocessen te modelleren omdat bedrijfsprocessen op een natuurlijke wijze als workflow kunnen worden voorgesteld. Workflowtalen kunnen echter voor meer dan alleen bedrijfsprocessen gebruikt worden. Elke logische opvolging van taken kan theoretisch door een workflowtaal gemodelleerd worden. Het kan zijn dat deze takenopvolging zeer complex is en dat er bijgevolg een aanpassing of toevoeging aan bestaande talen nodig is. Een van zulke complexe processen is terug te vinden in Computer Aided Engineering (CAE) en Multi-Disciplinary Optimization (MDO). Applicaties in deze domeinen verschillen zodanig in complexiteit dat ze onmogelijk door huidige workflowtalen kunnen worden voorgesteld. Dit probleem is een eerste maal onderzocht geweest bij SSEL (Software and Systems Engineering Lab, VUB). Hieruit vloeiden volgende probleemdefinities voort: *modularization of crosscutting concerns, advanced data support and advanced iteration support.* Meteen werd ook een conceptuele taal voorgesteld die een oplossing biedt voor de eerste twee problemen. Deze thesis stelt een mogelijke oplossing voor de derde en laatste probleemdefinitie, namelijk het ondersteunen van iteraties. Iteraties zijn algoritmen zoals optimalisatie methodes, Monte-Carlo methodes en vele anderen. In een eerste stap zullen we deze iteraties definiëren in een classificatie. In een tweede stap stellen we workflowpatronen voor die in het bestaande concept passen en die toelaten de gedefinieerde iteraties te implementeren. De conceptuele workflowtaal voldoet nu aan alle eisen en bijgevolg kan een eerste implementatie van de taal beginnen. Het wordt nu mogelijk om MDO applicaties te ontwikkelen met behulp van een echte workflowtaal. Ze krijgen hiermee alle directe voordelen die workflowtalen te bieden hebben: evolutie en hergebruik, aanpasbaarheid, samenwerking en gedistribueerde berekeningen.

Acknowledgements

My sincere thanks go to Prof. Dr. Viviane Jonckers and Niels Joncheere for giving me the opportunity to work on this project. They have provided me with the necessary assistance to reach the targets. The meetings that took place and the discussions we've held were always in a very constructive and communicative atmosphere.
I would like to take the opportunity to express my greatest thanks to three of my fellow students: Kim Bauters, Pieter Coucke en Wim Van Litsenborg. They have been of enormous importance in realizing this thesis as an evening student. Without their assistance during the past three years, I'd probably not even be writing this acknowledgement today.
Of course no man can really accomplish anything without the constant support of his loving family and girlfriend. Many thanks to my mother for both her emotional and financial assistance. Thank you Tine for your love, patience, understanding and support.

Contents

List of Figures

Listings

Chapter 1

Introduction

1.1 Context

Computer Aided Engineering is a specific virtualization technology that aids engineers in their daily tasks. CAE tools allow to build and analyse virtual product structures such as a building, a car, an airplane, etc. The goal of CAE is to improve the general product development cycle. Not surprisingly, this cycle is complex and consists of many subtasks and disciplines such as structural integrity, noise and vibration, systems dynamics and durability. Through CAE, these steps become simulated and automated and designs become optimized before the actual prototype construction.
However, there is more to it than just performing structural analysis on the design. After all, hundreds of different parameters might influence the virtual model. For example, consider we want to investigate the mass of car. A car has hundreds of different components and changing the thickness of one component influences the total mass of the car. This complexity asks for tools that abstract over these parameters and gain easier and faster insight in the design. The latter problem is defined as a specific niche in the market called PIDO [dPJ02], process integration and design optimization. These tools allow for design space exploration, multi-objective optimization and robust design techniques. These terms are explained in more detail in Chapter 3. Applications in this domain are often referred to as multi-disciplinary design optimization (MDO) applications [CNdPJ02].

In another domain we have workflow languages such as BPEL (Business Process Execution Language) [AA07] or YAWL (Yet Another Workflow Language) [Kne04]. They provide low level building blocks for modeling business

processes. BPEL is widely accepted in today's industry due to its maturity but has problems in standardizing its approach. YAWL on the other hand was developed as an answer on BPEL's problems and based on workflow patterns developed by van der Aalst [vdAtHKB02]. Through these patterns YAWL has a more fundamental formalization compared to BPEL.

1.2 Problem description

What is the relationship between workflow languages and Computer Aided Engineering? Looking at today's landscape of both domains, they seem fairly uncorrelated. On the one hand we see that CAE/MDO is in need for workflow systems. On the other hand, the market and most certainly the PIDO market show that this translation from engineering process to workflows is not in common practice. Paradoxically, the market wants it and needs it. One of the leaders in the PIDO market, called Optimus [Sol08], shows this demand. In this tool, the engineer can indeed express his process in some high level workflow.
So then why are MDO applications not using any of today's workflow languages? After all, by not using these languages they miss out on a lot of free constructs. Each time these applications introduce a new workflow feature, they reinvent the wheel. This paradox has been studied in recent research at the SSEL lab of the VUB ([JDVJ08]). The reason for the discrepancy in workflow need and workflow use by CAE has mainly three reasons, all caused by the complexity in requirements of MDO applications which are missing in today's workflow languages. First of all MDO workflows are in need for differentiating the main workflow concern from the secondary concern using the same workflow. Secondly, MDO workflows focus on data representation and manipulation and finally MDO workflows require advanced iteration constructs. The goal of the project can thus be formalized as the development of a workflow language leveraging current workflow approaches with modularization of crosscutting concerns, advanced data support and iteration constructs. These requirements are explained in more detail in Section 2.4

1.3 Solution description

In [JDVJ08] a conceptual framework is presented which builds upon the YAWL engine. It solves the problem of modularization of crosscutting con-

cerns by extending the sub-workflow construct. Secondly it proposes much better support for data handling. For example unlike other languages, data will be explicitly visualized in the workflow.
However the final problem for supporting iterations is yet unanswered in SSEL's research and language. This is exactly the topic of my thesis. We present a solution for capturing the advanced iteration constructs that MDO applications require.
The contribution has three parts: in Chapter 3 we present a classification of all MDO iterations from the viewpoint of a structural engineer. In Chapter 4 we present new workflow patterns suited for modeling these iterations and we map the iterations classified in Chapter 3 into these patterns. Finally in Chapter 5 we make a proof of concept, by implementating some of the MDO iterations using the proposed patterns.
The conceptual framework for modeling CAE workflows with the added support of advanced iteration patterns are of big importance to the CAE/MDO industry. It not only bridges the gap between the worlds, but it will support their software for evolution and reuse. By using this new language, MDO applications can focus on the actual structural engineering part. Not only does it offer support for modularization, data representation and advanced iteration, furthermore, it opens the doors for collaboration and customization due to the nature of the service oriented architecture of the framework such as native support for web services and parallelism.

Chapter 2

Literature review

2.1 Introduction

The goal of this thesis is to analyze the use of iterations in a new workflow language for CAE. In order to succeed in this, the different terms apparent in the goal statement need intensive research. Below, we will summarize the result of this prior research starting with a description of workflow languages, continuing in a description of Computer Aided Engineering and the current use of workflow languages in CAE. Finally we conclude the review section with the current status of the new language being developed at the VUB.

2.2 Workflow languages

At the heart of any successful organization is the drive to improve efficiency. Business processes are used to analyze this drive for efficiency: knowing how an organization works by identifying their processes and finding ways to improve the flow between them. We can summarize this statement in what is called Business Process Management, a methodology defined as "supporting business processes using methods, techniques, and software to design, enact, control, and analyze operational processes involving humans, organizations, applications, documents and other sources of information" [vdAHW03].
One way of capturing such business processes is through workflow languages, defined as software systems that manage and execute operational processes. Many such languages exist today such as UML activity diagrams, XPDL and BPEL. These approaches however lack a formal foundation, caused by the historical absence of a standardized definition of the core constructs of

business processes. Therefore a new approach for standardizing business processes has been introduced by Van der Aalst et al. [vdABtH+00], which is a pattern-based approach. These patterns are categorized into four classes. Below we will sketch the different pattern categories, and highlight some of the more commonly used patterns. Finally we will end the section on workflow languages by introducing YAWL, an open source workflow system that is implemented with van der Aelst patterns in mind.

2.2.1 Control-flow patterns

Patterns belonging to the control-flow perspective formalize tasks that make up processes and the relationship between them. In total there are 20 different patterns identified by the Workflow Patterns Initiative [vdAtH09], such as:

The sequence pattern, a pattern that describes a sequence between tasks and serves as the basic building block for processes (See figure 2.1).

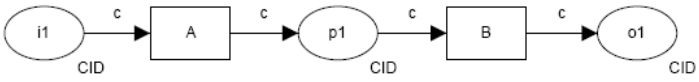


Figure 2.1: Sequence pattern

The exclusive choice pattern, a pattern that allows a thread of control to be directed to a specific task depending on the outcome of a preceding task as shown in figure 2.2.

Parallel split pattern, a pattern that allows for divergence of a branch into two or more parallel branches each of which execute concurrently, see figure 2.3.

Synchronization pattern, defined as the pattern that converges two branches into one, see figure 2.4. This pattern is usually preceded by the parallel split pattern.

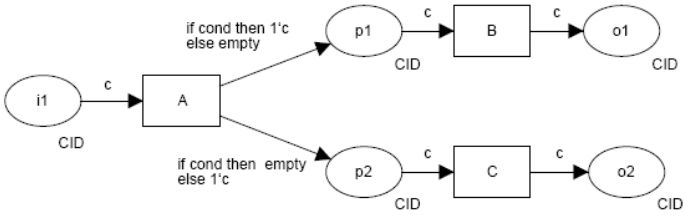


Figure 2.2: Exclusive choice pattern

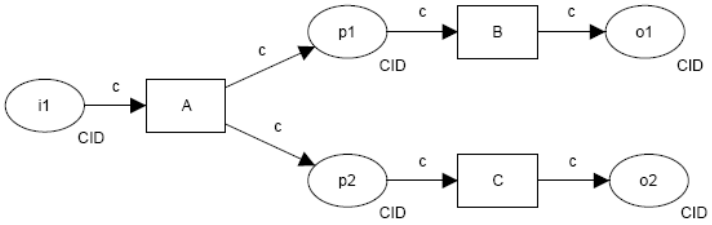


Figure 2.3: Parallel split pattern

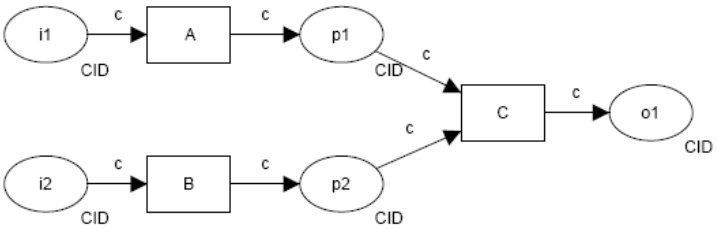


Figure 2.4: Synchronization pattern

2.2.2 Data-flow patterns

These patterns all handle data that is used, consumed, manipulated and transferred in some way during the flow of a process. These data patterns can be further subdivided into 4 categories: data visibility patterns, data interaction patterns, data transfer patterns and data-based routing patterns.

> **Data visibility patterns**, are patterns that control how data elements can be viewed or how data elements are scoped in a process. Typical scopes include task, block and global. There are in total 8 data visibility patterns.

Data interaction patterns, include those patterns that allow for communication of data elements between tasks. For example, patterns to pass data between two individual tasks (see figure 2.5) or patterns to communicate data between tasks and sub-processes (see figure 2.6).

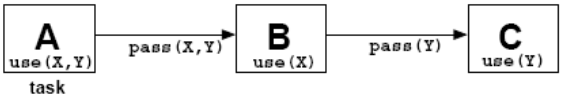


Figure 2.5: Task to task communication.

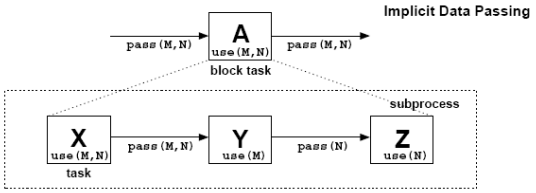


Figure 2.6: Block-task to sub-process decomposition.

Data transfer patterns, are patterns that consider the manner by which data can be transferred between different process components. Typical transfer patterns include: **Data transfer by value**, a pattern that avoids the need for shared memory; **Data transfer by reference**, used for accessing shared memory by utilizing its reference; **Data transformation pattern**, a pattern that allows data transformation to a data element prior to it being passed to a process component.

Data-based routing patterns, include those patterns that depending on the state of certain data elements will alter the control flow of the process. Typical patterns include: **Task precondition**, a pattern that allows a task to be preconditioned, meaning that the task can only proceed if the associated precondition evaluates positively; **Task post condition**, a pattern that allows a task to be post conditioned, which will ensure that a task cannot complete until specified output parameters exist and have been allocated a value.

2.2.3 Resource patterns and exception handling patterns

We decided to only shortly describe these two categories together in one section as these patterns are less important to our research. Resource patterns center around the modeling of human resources, for example to describe the relationship of specific tasks to the people in the organizational model. It allows for work distribution and effective work planning.
Exception handling patterns deal with the unexpected behavior or undesirable events that may be encountered during execution. These patterns should answer three questions: what to do with the work item that caused the exception, what to do with the other work items influenced by the problematic work item and what recovery action should be taken to resolve the exceptional situation.

2.2.4 YAWL: Yet Another Workflow Language

One of the major outcomes of the Workflow Patterns Initiative was the recognition of the need for a business process modeling language that would provide concrete implementations for the described workflow patterns. The result was the development of YAWL [vdAH03], [Hof03], Yet Another Workflow Language, a joint effort between Eindhoven University of Technology and Queensland University of Technology. YAWL is an open source initiative that exploits fundamental properties of Petri nets [LKD+08] and implements 19 out of the 20 control-flow patterns. YAWL consists of an operational environment (containing the workflow engine) and an editor that allows to visualize Petri nets and that allows to provide user interaction.
Figure 2.7 show the elementary building blocks available in YAWL. A workflow specification in YAWL is a set of extended workflow nets (EWF) which form a hierarchy in a tree-like structure. Tasks in the workflow can either be atomic tasks or composite tasks. The atomic tasks form the leaves of the tree, whereas the composite tasks refer to an EWF net at a lower level in the hierarchy. Each EWF net has exactly one input condition and one output condition.
To see how YAWL can be used in practice to reason about processes, we have added three examples that describe a process for booking a flight, hotel and a car followed by a task to pay for the bookings. The first example as seen in figure 2.8(a) demonstrates the multi-merge pattern, that is, upon each successful booking, the pay task is executed. In other words, the pay task is allowed to be executed three times. The second example as seen in figure

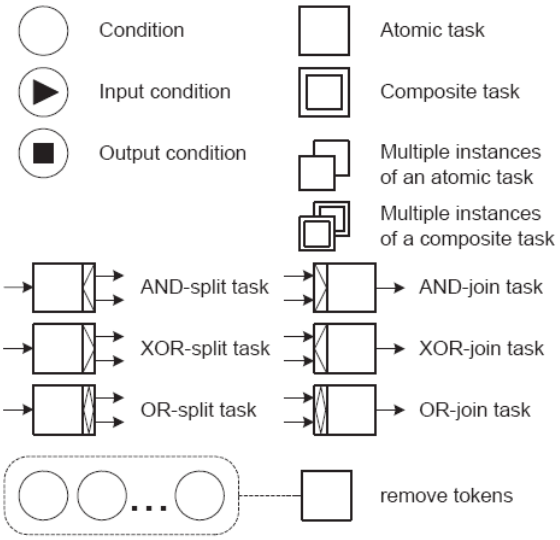
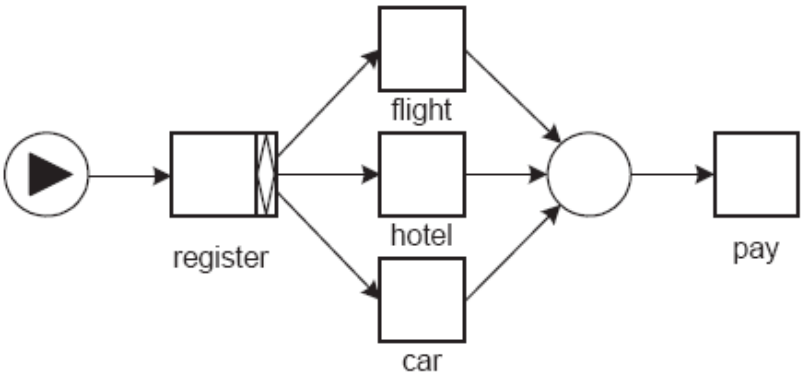
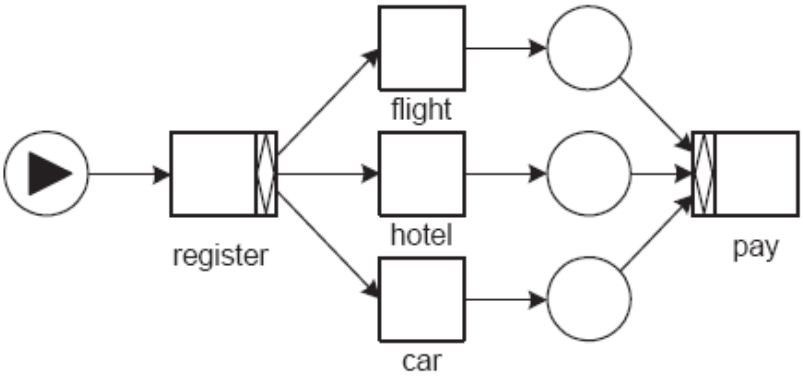


Figure 2.7: Symbols used in YAWL.

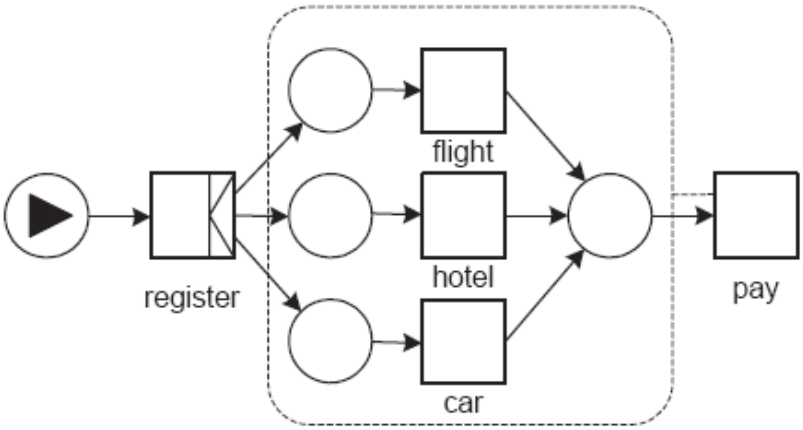
2.8(b) implements the structured synchronization merge pattern. This means that the pay task will execute only once, that is after all three preceded tasks have been successfully completed. And finally the third example as seen in figure 2.8(c) implements the discrimination merge pattern, meaning that the pay task is executed exactly once, but only immediately after the first task has been completed, canceling out the other two tasks.



(a) Multi-merge example.



(b) Structured synchronization merge.



(c) Discrimination merge.

Figure 2.8: YAWL in practice

The reason that we pay attention to YAWL is that it clearly shows, compared to other approaches, its intention for standardization and formalization. It wants to follow clear semantics and share it with the world by providing an open-source workflow engine. For this reason we will use the YAWL engine as a starting point for implementing the CAE workflow language. Although it is true that the current YAWL system does not yet provide implementations for all patterns, especially not the data-flow patterns, we see an active research group continuously improving YAWL. For example, research has begun in developing newYAWL [Min07], the next major change to YAWL that will provide implementations for all the currently existing van der Aalst patterns and provide a new set of control-flow patterns mostly related to more advanced iteration constructs.

2.3 Workflows for Computer Aided Engineering

2.3.1 CAD, CAE and PIDO

Computer Aided Design (CAD) is the use of computer technology to build virtual 3D geometric models. It provides a comprehensive description in shape elements, constraints and materials. CAD is used in many domains such as the medical society, gaming industry and automotive and aerospace industries. Example designs of the latter contain car engines and aircraft wings. However, CAD is limited in that it only informs us about the actual design form and fit. It does not provide any information about the functional performance of the design. Once the virtual model is created, engineers want to analyze the system and make the model undergo extreme conditions. Typical analysis consists in noise and vibration testing, acoustics, crash and durability. To meet these requirements, the engineering field of Computer Aided Engineering (CAE) has been developed that lift CAD applications to full Virtual Prototyping applications capable of shortening design cycles, reducing design costs and producing products with superior performance. All these functional performance tests are achieved through a methodology called the Finite Element Method (FEM), a numerical approximation process for solving Partial Differential Equations (PDE).
Engineers are also interested in getting answers to questions as: "What is the sensitivity of the performance to variations in the design properties?" or "What design features should be varied, and how, to achieve the target performance?" To answer these questions, PIDO or Process Integration and

Design Optimization comes to the rescue by automating design variation. Tools in this domain allow to model the analysis process and allow to iterate over this analysis process by intelligently choosing a set of values for a set of predefined parameters. Furthermore, designs can be optimized by minimizing and maximizing desired design parameters and finally they allow to automatically detect sensitivity points in the design.

2.3.2 Optimus

One of the leaders in the PIDO market is Optimus, a virtual design environment within which users can 'experiment' rapidly with the design of a product, and achieve an optimized design. It can do so by allowing to visualize the design and provide the engineer critical insights in the system's dynamics. This visualization is achieved through workflows. In this graphical network, the user can model his analysis process, define the inputs and boundaries, outputs and constraints. Once the structure of the process is thought to Optimus, the environment will automatically generate, analyze, explore and track the design alternatives, launching a series of 'experiments' in a 'virtual test laboratory' (see figure 2.9) through systematic evaluation of the responses for each virtual run.

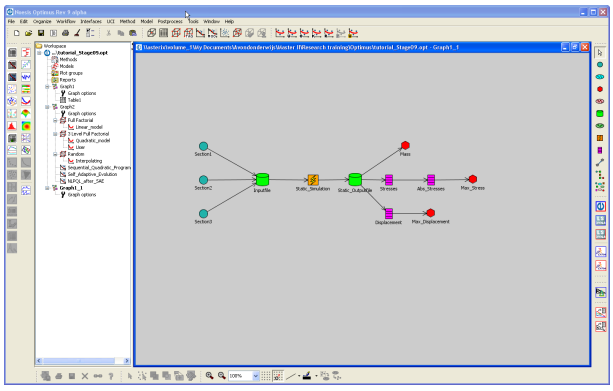


Figure 2.9: Optimus virtual test laboratory.

Pennsylvania bridge

The best way to show how Optimus works is through a practical approach. In the following subsections, we will demonstrate Optimus by capturing the

process for prototyping the Pennsylvania bridge and investigating several functional performance attributes. Figure 2.10 shows the structure of the bridge and the design inputs we are interested in. The bridge consists of several connected beams divided into three materials. We have the upper beams (material 1), the beams forming the deck (material 2) and the inner beams (material 3). A small Finite Element Method program has been developed specifically designed to handle the structure of this bridge analyzing several attributes as mass, displacement and stress. We will choose to investigate these attributes by varying over the cross-section areas of the three sets of beams. The FEM program is parameterized in a file-based approach:

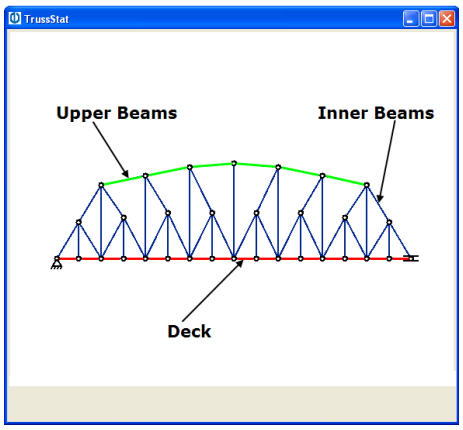


Figure 2.10: Pennsylvania-bridge type structure.

it expects a file describing the values for the different design variables and responds with an output file describing the resulting values for the performance attributes. For a complete description of the structure of these two files, we refer the user to Appendix A.

Optimus workflow

Optimus provides the user the possibility of modeling a process through an Optimus workflow consisting of several data and activity elements. Below, we will describe the different Optimus workflow elements and show how they can be used to model the analysis process of the Pennsylvania bridge. The resulting workflow will look as in figure 2.11. We will provide a short description for each workflow element in order of appearance.

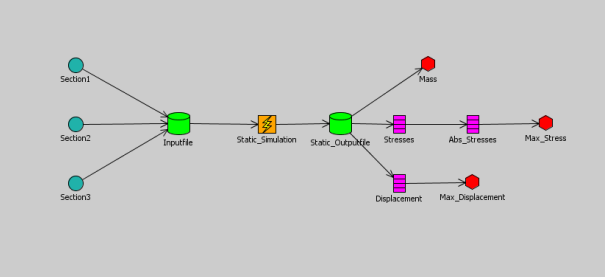


Figure 2.11: Optimus workflow

Input element An Optimus input element is a data element corresponding to a design input of the structure to be analyzed. The workflow in our example shows three inputs, corresponding to the three sections of the Pennsylvania bridge. During execution of the workflow, Optimus will assign values to these input variables. These values will always be in the range defined in the input element definition.

Input file element The three input sections are connected to an input file, which is another data element representing a file on disk. This input file will later be used as the first parameter in the external FEM program. In our example the input file will be the file as displayed in Appendix A were during workflow execution we will substitute the three material cross-section area values with the three current input values.

Action element An action element is an activity element responsible for executing external processes. In our example we will simply call the FEM executable and pass it the 2 connected files: the input file and the output file. During execution, Optimus will call the external executable, pass as parameters, the physical file paths of the input file and output file. The external executable will open the file on disk, execute accordingly and write its result into the output file. This file is then captured by Optimus into an output file element.

Output file element An output file element is another data element responsible for capturing the result of an external program into a file and extracting generated values into Optimus output and or vector elements. In our example, the FEM program will generate a file as shown in Appendix A and extract one output value corresponding to the mass of the bridge and two

output vectors, one representing the different stresses and one representing the different displacements.

Output element An output element is a data element holding one scalar value. An output definition can further be extended to hold information about low and high constraints or about reaching a certain target. This information can later be used for example by optimization algorithms that will try to minimize or maximize that output within the given constraints or that will try to get the output as close as possible to the given target value. Our workflow defines three outputs: one to define the mass of the bridge (that will retrieve its value directly from the output file), one output that represents the maximum of all the extracted stresses and one output that gives us the maximum value of all extracted displacement values.

Output vector element An output vector element is a data element holding multiple values at once. In our example we see three vectors, although only two of them hold information about different kind of functional performance attributes. The first holds the information for the extracted displacements values, whereas the second vector holds the information for all the extracted stresses values. Furthermore, we see a third vector that manipulates its incoming vector by making all vector values absolute.

Design iterations

Once we have completely defined our Optimus workflow, we can start exploring our design space trough the different iterations available. An Optimus design iteration is a loop of virtual experiments where an experiment is defined as one execution of the workflow with a unique set of values fed to the input elements. All experiments will be collected so that after the iteration, post processing is possible on the executed experiments. A complete classification of these iterations will be given in Chapter 3 where we will use the Pennsylvania bridge as well to exemplify the theory.

Post-processing

To give a meaning to the executed experiments, Optimus provides an extensive set of post-processing features, each of them designed to extract as much information as possible about the executed iteration. Figure 2.12 shows some of the available plots. A summary plot shows a flat list of all executed

experiment values with corresponding input and output values. A scatter plot (either in 2D or 3D) shows in a graphical way, the correlation between pairs of design variables. A correlation plot denotes the numerical correlation coefficients between design variables. A 3D plot draws a surface plot of a selected output variable in function of two selected input variables with isolines projected on the bottom. And finally, an optimization progression plot draws the progress of reaching a specified goal during the different executed iterations.

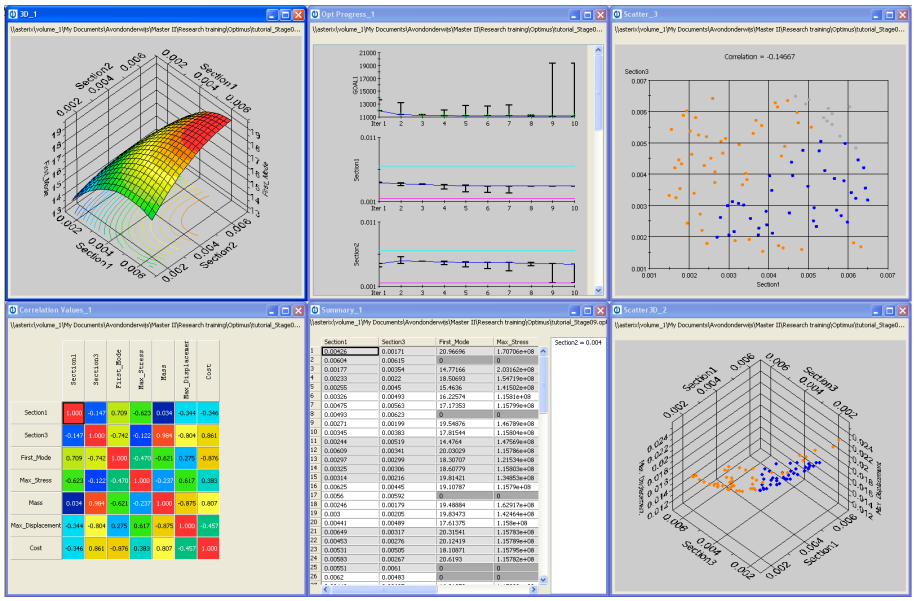


Figure 2.12: Optimus post-processing. Top left: 3D plot, Top middle: Optimization progress plot, Top right: 2D scatter plot, Bottom left: Correlation plot, Bottom middle: Summary, Bottom right: 3D scatter plot

2.4 Current Status

From the literature study of MDO applications we can now deduct the requirements that a suitable workflow language would need. Below we will summarize these requirements and explain the current solution offered by SSEL's conceptual workflow language.

2.4.1 Requirement 1: Separation of concerns

In CAE applications we have many concerns, where some are more important than others. A main concern could be to minimize an objective, whereas side concerns could be licensing and logging information. In software engineering, this problem is referred to as the **tiranny of the dominant decomposition** [TOH+99]. A concern on its own can be separately modelled into sub-workflows. However connecting sub-workflows is up until now only possible in a one dimensional way, therefore causing secondary concerns to be scattered accross the workflow. To remedy this problem, SSEL has introduced **control ports** that allow, in tradition with aspect-oriented techniques, inversion of control. Control ports are the entries of a task or complete workflows and have attached connectors that can be annotated with inversion of control actions. This allows for redirection and resume from and to specified tasks.

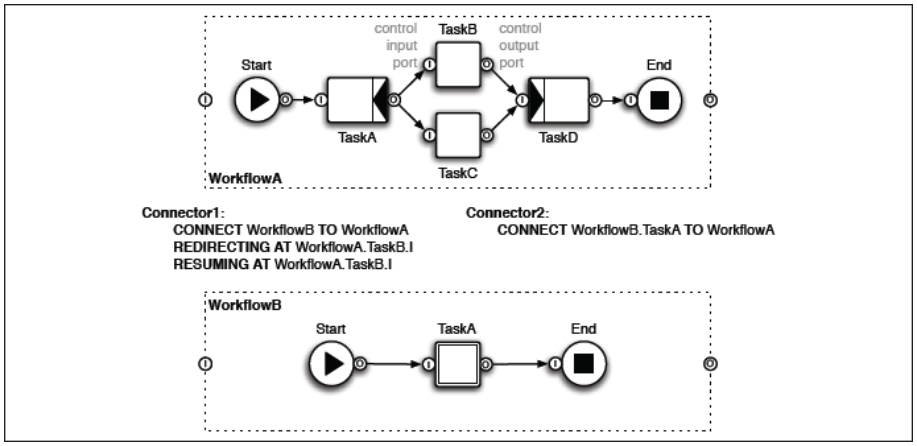


Figure 2.13: Redirecting workflow A to Workflow B.

Figure 2.13 shows an example of this construction. We see how the connector connecting Task A to Task B is annotated with a redirection statement that will cause Workflow B to be executed before Task B.

2.4.2 Requirement 2: Advanced data support

A second requirement is related to the extensive data use within MDO applications. In the description of an Optimus workflow, of all the discussed items, at least five out of six are related to some usage or manipulation of data. The input, output and vector elements are pure data representation elements, whereas the input file and output file element manipulate and

transfer data. Although there are several data patterns available (see Section 2.2.2), the problem is that all these patterns are tightly coupled with the control-flow perspective. Even worse, none of the data-flow patterns have up until today been implemented in YAWL.
SSEL's workflow approach resolves this issue by introducing **data ports** and first-class **data flow** constructs. Data ports represent the input and output parameters of a task and workflow, whereas a data flow construct transfers data from one port to another while transforming the transferred data by the specified manipulation task and could be used for example to extract values from a vector element. This way, all van der Aelst data patterns can be expressed with the added benefit of being explicitely visualized into the workflow as seen in figure 2.14.

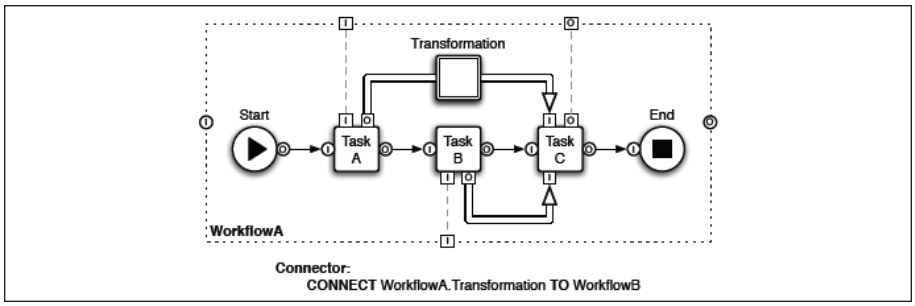


Figure 2.14: Data ports and data transfer.

2.4.3 Requirement 3: Advanced iteration support

Iterations are still an open issue for SSEL's language. In Chapter 3 we address the iterations that are used inside MDO applications. To make a workflow language useful for MDO applications, it needs to provide iteration patterns that allow to abstract over all classified iterations. Looking at today's workflow language, none of them come of even near this demand. YAWL supports the multiple instance pattern allowing a predefined number of instances to be created. However, many times, MDO iterations such as optimization algorithms are non-deterministic, making this pattern unusable. BPEL comes closest related to iteration support by providing a for-each loop and a while loop. Indeed, this is exactly the reason, we choose BPEL as reference language to model the iteration patterns in Chapter 4. In a later stadium, as YAWL is still the preferred language, our proposed patterns will have to be translated into YAWL. This will be possible as looking at the specificiations

of the upcoming language newYAWL [Min07], new iteration patterns will be added therefore supporting all constructs that BPEL supports today. Furthermore, a CAE iteration is not the same as a simple for-each or while loop. They will have to be leveraged with iteration data and several placeholders to make them really usable.

Chapter 3

CAE Iterations

3.1 Introduction

This section involves a thorough investigation and resulting description of the technologies in the PIDO market. The objective is to explain the different iterations available. We will focus on the actual iterations divided in three classes each described in its own section. We believe this description is important for two reasons. First of all, the final goal is to develop a workflow language specifically designed to aid the CAE/PIDO market. As such it is important to collect all information from this domain as to be sure that the resulting language can capture all methodologies currently available. Secondly, the theory behind tools such as Optimus is an engineering field not directly known to the software engineering field. We hope that our sidestep into this domain will guide future researchers so that no time is lost in the next phase of the project in developing a workflow language for CAE.

3.2 Design exploration iterations

Product engineers require tools to understand rapidly their design space. They want a way of structurally investigating the influence of the design input parameters on the output responses of the design. These investigations can take place either manually or automatically.

3.2.1 Table method

A table method is a type of design space exploration iteration that allows the user to manually define his set of experiments. The user will choose himself the number of experiments and assign himself the values for each input and for each experiment. As a result a table method always yields an input matrix with row dimension equal to the number of defined experiments and column dimension equal to the number of inputs.

3.2.2 Design of experiments

Design of experiments (DOE) [LA93] is a systematic approach to get the maximum amount of information out of various types of experiments while minimizing the number of experiments. Think of a DOE as a detailed experimental plan, studying the influence of design factors on the design responses. The way this experimental plan is built, is what makes one type of DOE differ from another. But whatever the plan, they all share as primary goal to extract the maximum amount of information ,regarding the factors affecting a production process, from as few observations as possible.
Since executing experiments can be a time-consuming task, further exploration is done trough Response Surface Models [MMAC08], a mathematical approximation methodology that predicts values based on the experiments calculated by the DOE.
There exist many kind of DOE methods where all can be classified into two categories: Orthogonal designs and Random designs.

Orthogonal designs: The experimental plan is built by selecting orthogonal points in the design space. As such the factors in an experiment are uncorrelated and can be varied independently of each other. Widely used methods are fractional, full-factorial designs, central composite designs and Box-Behnken designs. The disadvantage of these methods is the potential danger of creating many useless experiments if one of the factors turns out to have no influence on the design.

Random designs: The points in the design space are chosen based on a random process. The most common used random DOE is the so-called Latin Hypercube Design (LHD). Random DOE methods do not create useless experiments as if one factor appears unimportant, the others will still give information on the responses.

For clarifying the topic on DOE methods, below we highlight two of the more commonly used DOE methods: Full factorial methods and Latin Hyper Cube methods.

Full Factorial Design

In a full factorial design, every setting of every factor appears with every setting of every other factor. As such the effects of multiple factors are investigated simultaneously and the effects of each factor are independent of the remaining factors. The number of experiments in a full factorial design can be calculated through the formula

$$N = \prod_{1}^{k} n_i, i = 1, ..., k \tag{3.1}$$

where N is the number of required experiments, k the number of factors and n_i the number of levels for a certain factor i.

Two Level Full Factorial In a two level full factorial, the level of all input factors is set at 2 and as such the number of experiments for this type of DOE is 2^k. The DOE method will automatically assign the extreme values of the input domain to a factor. If we were to sample 3 input factors, we can visualize the design of this experimental plan as a cube as shown in figure 3.1.

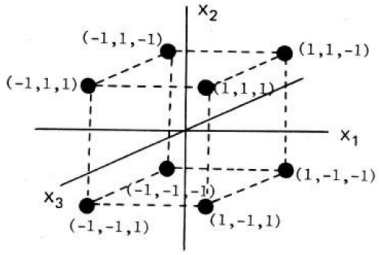


Figure 3.1: Two level full factorial design for 3 input factors.

A two level full factorial is not that powerful in that it is only capable of sampling linear mathematical models as in

$$Y = \alpha_0 + \sum_{i=1}^{k} \alpha_i X_i \tag{3.2}$$

.

Three Level Full Factorial For sampling more complex mathematical models such as second order models, we need more levels, at least 3. Consequently quadratic models as presented in the following formula

$$Y = \alpha_0 + \sum_{i=1}^{k} \alpha_i X_i^2 + \sum_{i=1}^{k} \alpha_{ii} X_i^2 + \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \alpha_{ij} X_i X_j \tag{3.3}$$

are ideally modeled with a three level full factorial. Figure 3.2 and 3.3 visualize the layout in the design space for a three level full factorial of 2 respectively 3 factors.

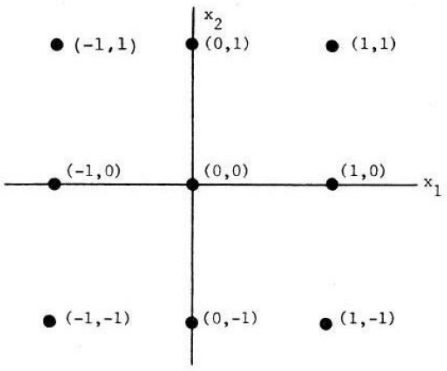


Figure 3.2: Three level full factorial design for 2 input factors.

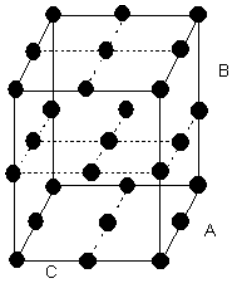


Figure 3.3: Three level full factorial design for 3 input factors.

Adjustable Full Factorial In several scenarios, the above DOE methods will actually perform too many experiments. Take for example the following formula:

$$Y = \alpha_0 + \alpha_1 X_1 + \alpha_2 X_2 + \alpha_3 X_1^2. \tag{3.4}$$

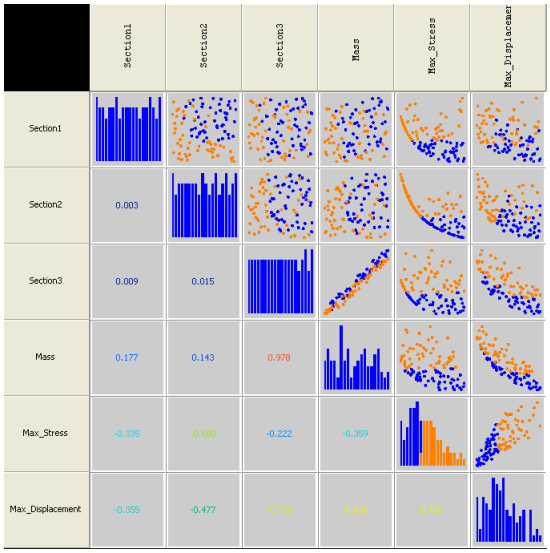
Since this equation is of second order, we might argue that we need a three level full factorial resulting in $3^2 = 9$ experiments. However, careful readers notice that only factor X_1 has a quadratic effect on the response whereas factor X_2 has just a linear effect. Fortunately and also deductible from equation 3.1, we are allowed to assign different levels to different factors. In this case we set factor X_1 to level 3 and factor X_2 to level 2 resulting in just 6 experiments.

Latin-Hypercube design

A Latin-Hypercube DOE method belongs to the category of random methods, that is the design points are chosen based on a random process. However, we do not want our design points to be completely random, as this can lead to a design filled with clustered points, which is not an interesting situation for exploration purposes. Instead we want points that are as much space-filling as possible. In statistical sampling, structured randomization is achieved through a Latin square, which is defined as a grid containing sample positions, if and only if there is only one sample in each row and each column. A Latin-hypercube DOE will generalize this idea in multiple dimensions by forcing only one sample on each axis-aligned hyper plane.

In practice

Let us take a look at how exactly a Design of experiments can help us in exploring the design. We will apply a Latin-Hypercube experimental plan of 100 experiments on the Pennsylvania bridge case in order to get an idea how the three sections influence the different performance attributes. The complete summary of results is listed in Appendix B.1. Figure 3.4 shows two interesting plots for the resulting Latin-Hypercube. On the left (3.4(a)) we have a graphical matrix overview mainly in scatter formation, giving us information about the influence of each design variable on any other. On the right hand side (3.4(b)) we have the numerical equivalent giving us a correlation value between -1 and 1. Reading or viewing through the different cells of the plots, we can deduct important information such as linearity between inputs and outputs. For example, we clearly see how *section 3* has a



(a) Correlation scatter plot

	Section1	Section2	Section3	Mass	Max_Stress	Max_Displacement
Section1	1.000	0.003	0.009	0.177	-0.335	-0.355
Section2	0.003	1.000	0.015	0.143	-0.680	-0.477
Section3	0.009	0.015	1.000	0.978	-0.222	-0.729
Mass	0.177	0.143	0.978	1.000	-0.359	-0.830
Max_Stress	-0.335	-0.680	-0.222	-0.359	1.000	0.766
Max_Displacement	-0.355	-0.477	-0.729	-0.830	0.766	1.000

(b) Correlation values plot

Figure 3.4: Latin-Hypercube plotting for the Pennsylvania bridge case

strong linear relation with performance attribute *mass* and a somewhat less (-0.729) linear relation with *maximum displacement*. *Section 2* has an influence on both *maximum stress* and *maximum displacement* whereas *section 1* compared to the other two inputs, has almost no influence at all on the three outputs.

3.3 Numerical optimization iterations

A second discipline in design automation is the search for optimality in the virtual design. Optimization is an active research area not just in engineering, but in economics and mathematics as well. The objective of optimization is the search for extrema that are apparent in the model, once or many times, locally or globally. In Optimus, intelligent optimization algorithms seek automatically the optimal value for one or many manually specified objectives. The objective can be to minimize or maximize a function satisfying a number of constraints. When a feasible solution is found, the objective function is called an optimal solution. With these tools, an engineer is capable of e.g. studying how to minimize the mass of a car given a range of variables with specified constraints.

Optimization algorithms can be classified by number of objectives: **single-objective optimization** or **multi-objective optimization**, and by search scope, resulting in either **local optimization methods** or **global optimization methods**. As with the description of DOE methods, we will only highlight the most commonly used algorithms.

3.3.1 Single-objective optimization

Single objective optimization methods have only one objective. As such, the goal of single-objective optimization is to find an optimum point within the specified domain satisfying a number of constraints.

Local optimization methods

Local optimization methods only use the local information, such as gradient information, from the objective function to search for extrema. As such, the global structure of the objective function is unknown to the local method. The advantage of this approach is that it has a fast convergence time. The disadvantage however, is that the property of locality causes only local extrema to be found, which might not be an extremum for the global function. We say that the algorithm gets trapped into local minima or maxima.

Sequential Quadratic Programming This algorithm finds one optimal local solution for non-linear constrained problems. The method is an approximation search algorithm based on Newton's method with the added possibility of satisfying constraints. The latter is done with the mathematical theory of Lagrangian. The basic idea of the method involves in formulating a quadratic programming sub problem in each iteration which is obtained by linearizing the constraints and approximating the Lagrangian function. The quadratic sub problem is than solved using Newton's method. For a more detailed description we refer the user to [Sto85] and [P.93].

Global optimization methods

Global optimization methods try to find the absolute best point in the objective function. Not surprisingly, since an objective function can have many local extrema, finding a global optimum is a lot more challenging. Global optimization algorithms have the advantage not to get trapped in local extrema like local optimizations do, but this comes with an expensive cost of

having to perform a big amount of experiments. It is therefore advised to use global optimization algorithms only when there is little knowledge about the design space. Heuristic search methods, such as evolutionary algorithms and simulated annealing, are accepted as the best solution to global optimization problems.

Evolutionary algorithms [Bey98] are heuristic optimization algorithms inspired by biological evolution. Indeed to find an optimal solution, EA algorithms will use genetic operators such as reproduction, mutation, recombination and selection. These operators are applied to an initial population of solutions evolving into an optimal solution.

3.3.2 Multi-objective optimization

In many cases, we will have more than one objective to minimize or maximize. These objectives can be conflicting; therefore a trade-off between the criteria is necessary to ensure a satisfactory design. Depending on the number of suggested solutions, we can further subdivide multi-object optimization algorithms in those that generate a so-called Pareto front (see further); generating a set of feasible objectives and those algorithms that present one compromised solution between the different objectives.

Pareto-front methods

A Pareto front (see figure 3.5) is the border continuous line plotted in the objective function space denoting possible optimal combinations of objective values. Decreasing the value of one objective while keeping the other one constant, will move the design point into the infeasible domain, whereas increasing the objective while keeping the other one constant, will make the design point non-optimal.
Methods in this category will therefore generate a Pareto front by assignment of weight combinations to the objectives. One algorithm will mainly differ by another in the way the Pareto front is constructed.

Compromised methods

Instead of generating a Pareto front, multi-objective optimization algorithms in this category will find one compromised solution. There are different ways

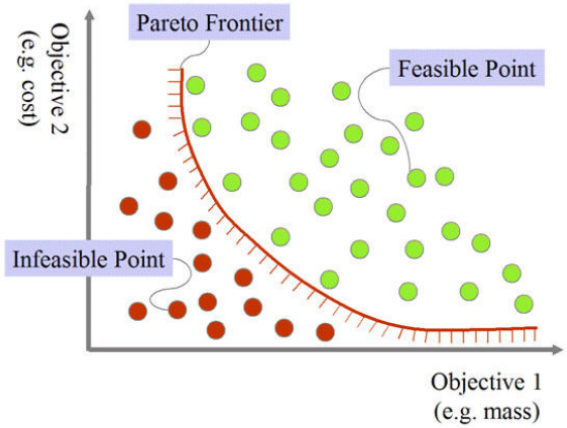


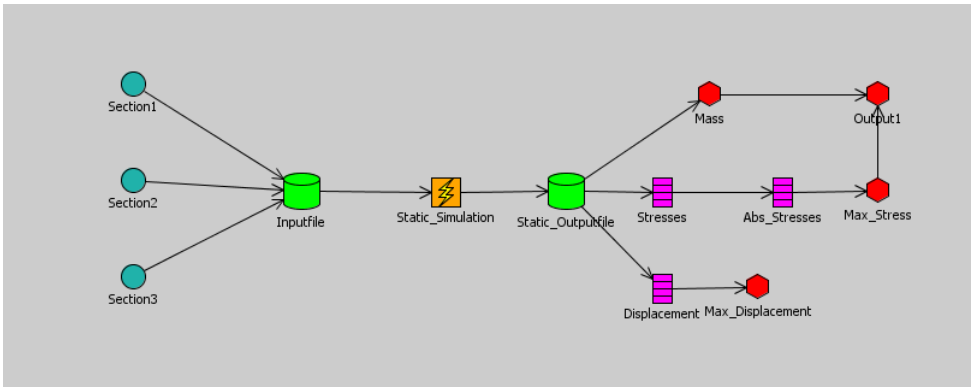
Figure 3.5: Pareto front.

to achieve this. For example a **trade-off method** will convert a multi-objective optimization problem into a single objective optimization problem by selecting one objective as primary objective while treating the other objectives as constraints. Another possibility is to let the user rank the objectives by importance as is done in the **hierarchical method**. Each objective function is then minimized individually but with the added constraint that the new minimum cannot exceed a prescribed fraction of the previous objective function.

3.3.3 In practice

Again to exemplify the theory, we experimented with Optimus and performed three optimizations on the Pennsylvania bridge. We have slightly adapted the workflow to include a fourth output which we will call *Cost* and that will be depending on *Mass* and *Maximum stress* with the formula: Cost=1.89*Mass+4.4e-5*Max_Stress. The adapted workflow is depicted in figure 3.6(a). Our objective was to minimize *Cost* as much as possible. This minimization is achieved by applying three optimizations on it. We started with a local sequential quadratic programming optimization (SQP). The resulting value as seen in the optimum plot (figure 3.6(b)) was $1.110E04$. The optimization needed in total 55 experiments as is seen in Appendix B.2.1 and in the optimum plot in the second column's caption. Then we applied a second optimization with the same objective, only this time using a global self adaptive evolution (SAE) which is a genetic algorithm. The resulting value

as seen in the optimum plot in figure 3.6(c) has been lowered to $1.099E04$. However it took the algorithm 349 experiments to get to this minimal value as seen in Appendix B.2.2. Finally we applied again a local sequential quadratic optimization algorithm, only this time starting from the input values that generated the optimal value for the SAE algorithm. These values are for section 1,2 and 3 respectively 3.103E-03, 4.011E-03 and 2.505E-03. The idea is to see if we can further fine-tune to an even smaller value. And indeed, after having performed an additional 14 experiments (see Appendix B.2.3), SQP resulted an optimal value of $1.098E04$ as seen in figure 3.6(d).



(a) Adapted workflow Pennsylvania bridge case with added output *Cost*

Optimization method = Sequential Quadratic Programming
Objective = Minimize Cost

	Start	End [55] (25)	Low	High
Inputs				
Section1	0.004	0.00337	0.0015	0.0065
Section2	0.004	0.00439	0.0015	0.0065
Section3	0.004	0.00273	0.0015	0.0065
Outputs				
First_Mode	18.21856	19.23955	18	
Max_Stress	1.158e+08	1.06966e+08		1.5e+08
Mass	4535.12195	3380.82013		
Max_Displacement	0.01478	0.01804		
Cost	13666.57413	11096.23935		
GOAL	13666.57413	11096.23935		

(b) SQP Optimum plot for Pennsylvania bridge case resulting in Cost optimal value of $1.110E04$

Optimization method = Self-adaptive Evolution
Objective = Minimize Cost

	Start	End [349] (349)	Low	High
Inputs				
Section1	0.004	0.0031	0.0015	0.0065
Section2	0.004	0.00401	0.0015	0.0065
Section3	0.004	0.00251	0.0015	0.0065
Outputs				
First_Mode	17.98211	18.80821	18	
Max_Stress	1.158e+08	1.16479e+08		1.4e+08
Mass	4535.12195	3101.50406		
Max_Displacement	0.01478	0.01967		
Cost	13666.57413	10986.92622		
GOAL	13666.57413	10986.92622		

(c) SAE Optimum plot for Pennsylvania bridge case resulting in Cost optimal value of $1.099E04$

Optimization method = NLPQL
Objective = Minimize Cost

	Start	End [81] (54)	Low	High
Inputs				
Section1	0.0031	0.0031	0.0015	0.0065
Section2	0.00401	0.00398	0.0015	0.0065
Section3	0.00251	0.00251	0.0015	0.0065
Outputs				
First_Mode	19.25334	19.24386	18	
Max_Stress	1.16479e+08	1.16372e+08		1.4e+08
Mass	3101.50086	3100.09159		
Max_Displacement	0.01967	0.0197		
Cost	10986.92563	10979.53456		
GOAL	10986.92563	10979.53456		

(d) SQP Optimum plot for Pennsylvania bridge case (starting from optimal SAE value) resulting in Cost optimal value of $1.098E04$

Figure 3.6: Optimization in practice

3.4 Robustness and reliability iterations

3.4.1 Introduction

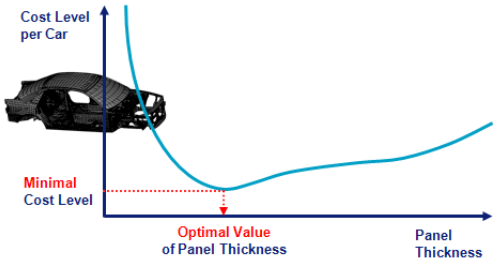
After having explored the design space and after hunting for optimality, we need to evaluate the robustness and reliability (R&R) [NdPJS03] of the design. The problem is in a way a bit cynic, that is, by trying to speed up the design process through virtual prototyping and simulation, we introduce possible problematic behaviors. The reason is of course that CAE can only simulate reality to a certain extent. Our virtual model differs by some degree with the model in reality. After all, we use techniques such as meshing where we discretize the design. As such a simulation output will always yield the same result given the same input values. In real life however, there is a certain variability on parameters where each same value might give a different result. This can have different reasons; take for example environmental condition influences or structural degradation (ageing, fatigue, damage ...).

Robust design is then defined as the discipline to study the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environment conditions. We want to study the sensitivity of outputs to inputs variability. A useful measurement for robust design will be the standard deviation.

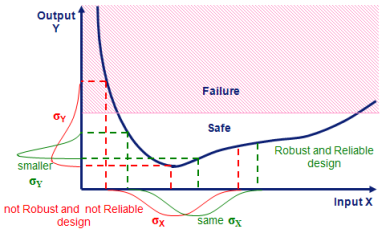
Reliability analysis is the discipline that aims to define the probability that a failure is attained as a result of input variability. This can be measured by the failure probability and the reliability index.

Figure 3.7 shows in more detail how R&R works. Figure 3.7(a) depicts a scenario for a car where we study the influence of the thickness of the car's panel in function of the total cost of that car. We notice that if we take a very small thickness that our cost will rise significantly due to the resulting production of an instable car, which will lead to lowered sales, law suits and company image loss. Taking our thickness too high, will lead to increased manufacturing costs as we need more material, increased use of gas etc ... In figure 3.7(b) a rasterized zone is depicted to highlight that any cost that falls into this zone is inacceptable. If we now take a first variability into account for our panel thickness called σ_x as seen in figure 3.7(b) in red and analyze the resulting σ_y it teaches us that the standard deviation is too big and that furthermore there is an inacceptable risk that some of the values fall in the

forbidden failure zone. We say that the design is neither robust nor reliable. Then by simply moving the distribution to the right on the x-axis, we reveal an interesting result as seen in green in figure 3.7(b). Suddenly the standard deviation became much smaller and none of the values fall into the forbidden zone. The design has become both robust and reliable.



(a) Cost function: x-axis = panel thickness, y-axis = cost of car



(b) Robustness and reliability

Figure 3.7:
Robustness and reliability study on cost level per car in function of panel thickness.

3.4.2 Monte-Carlo Method

One of the more popular techniques for the study of robustness and reliabilty is the **Monte-Carlo Method**: a sampling method that performs a number of simulations with randomly selected parameter combinations for the given distribution and calculates the standard deviation through formula 3.5.

$$\sigma_y = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(y_i - \overline{y})^2} \tag{3.5}$$

As mentioned, the standard deviation is a measurement for the study of robustness only. To study the reliability we need the probability of failure. This can be easility calculated through a Monte-Carlo method through formula 3.6, where P_f is the probility of failure, N_f denotes the number of failed samples and N the total number of trials.

$$P_f = \frac{N_f}{N} \tag{3.6}$$

3.4.3 In practice

We will evaluate the robustness and reliability of the stress on a single beam (beam 28 to be precise) by incorporating a variability on section 1. Our new workflow will look as in figure 3.8.

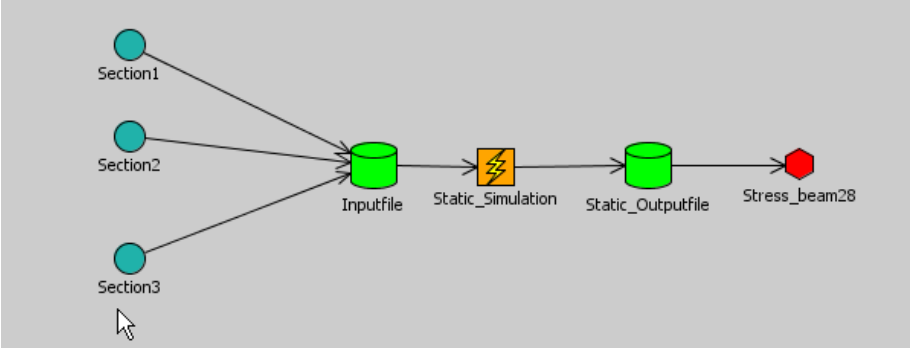


Figure 3.8: Workflow for robustness and reliability study on beam 28.

To put a variability on section1, in Optimus, we can assign it a distribution. In this example we will assign a normal distribution with a low bound of -0.0001. This distribution is depicted in figure 3.9.

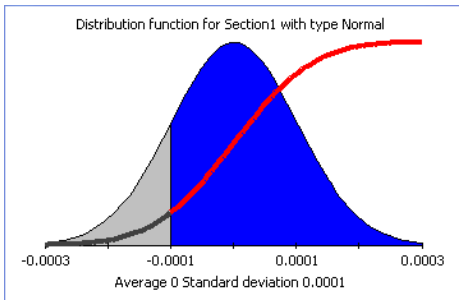


Figure 3.9: Normal distribution on section 1 with low bound -0.0001.

Finally we need to put a constraint on beam 28, if we want to study its reliability. For this example we choose a high constraint of 1.2E8. Now that the workflow, distribution and constraints are set up, we can apply a Monte-Carlo method. The resulting experiments are displayed in Appendix B.3. Assessment of robustness and reliability is now achieved by post-processing the Monte-Carlo method with Histogram plots. .
First of all notice in figure 3.10 how the resulting distribution on section 1 is

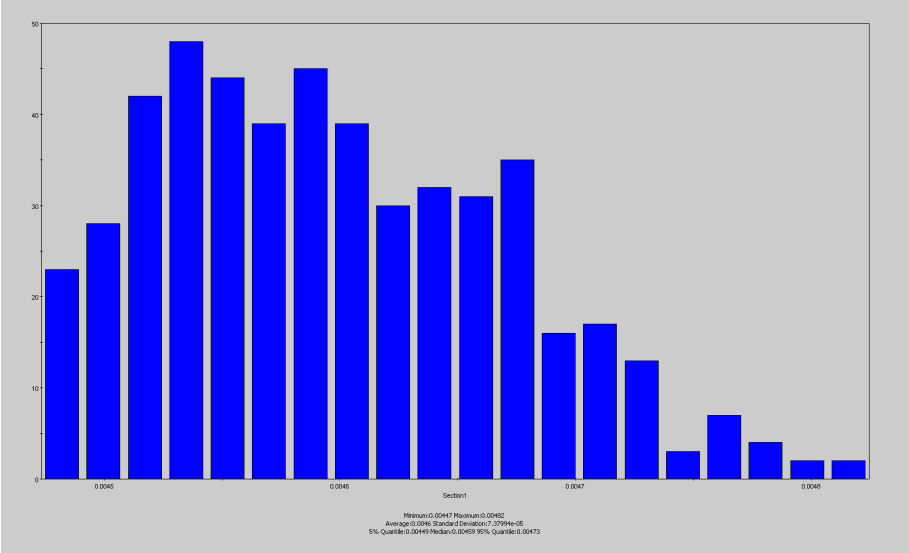


Figure 3.10: Histogram plot for section 1

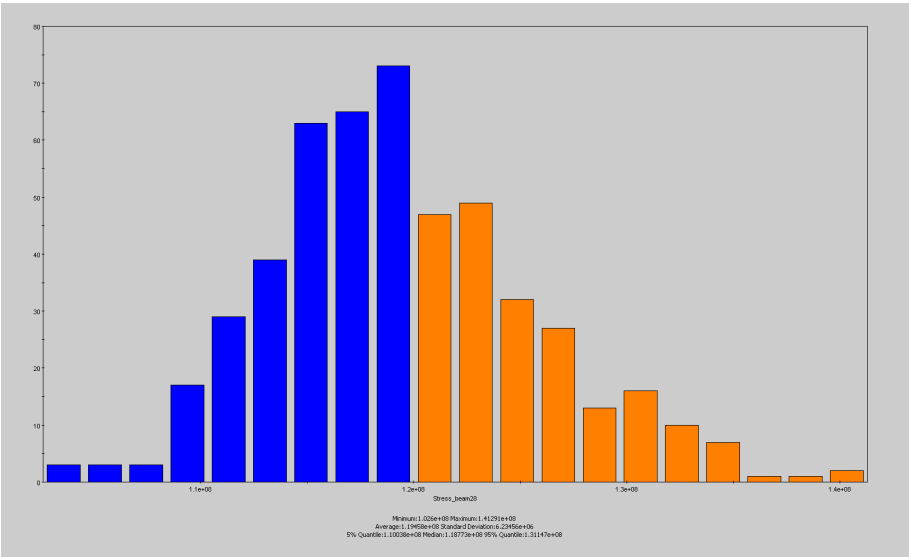


Figure 3.11: Histogram plot for beam 28

indeed conforming to our specified normal distribution with low bound. From figure 3.11 we see that our standard deviation is 6.23456E06 and our mean value is 1.19458E08. Given the closeness in magnitude of both values, we can deduct that the response distribution is too widespread and hence

our design is not robust. Furthermore the orange bars in the histogram all show experiments response values in the infeasible domain. Given the huge amount (almost 50%) of infeasible experiments, we can conclude that our design is not reliable.

Chapter 4

CAE Patterns

4.1 Introduction

Now that we have a good idea of what kind of iterations the CAE domain has to offer, we need to analyse these iterations for commonality and variability, thus defining abstractions and creating patterns. We translate the iterations as seen from the structural engineer into patterns as seen by the software engineer. As such a pattern suitable for modelling an iteration should be seen just like a classical design pattern like those from the gang of four [GHJV94]. We provide elementary building blocks in a workflow system built upon the existing workflow elements, that allow to construct an iteration independent on the type of iteration. To give an example, the structural engineer can use our pattern to construct a random design of experiments iterating over the analysis.

4.2 Pattern definition

A pattern will be uniquely defined by its structure and internal iteration data. Below we have listed the properties that a pattern structure has to comply to.

4.2.1 Pattern structure

Below, we present a number of properties that together uniquely define each pattern. These properties will return in the description of a pattern.

Property 1 - External versus internal iteration structure: We call the external structure of an iteration the construct of repetition whereas the internal structure is the construct within one step of the iteration. This is comparable to the structure of loops within programming languages, where the external structure of a while loop is the statement inside the *while header*, whereas the internal structure is the *while body*.

Property 2 - Abstraction level: A CAE pattern will provide abstraction both on the external structure as on the internal structure. The external structure abstracts over a class of iterations such as optimizations or design of experiments, whereas the internal structure allows to abstract over the type of iteration in one class. The latter is done by providing specific placeholders inside one abstraction. For example a design of experiments differs from an optimization in external structure, hence the user will be using two different patterns, while a self-adaptive evolution optimization differs from a sequential quadratic programming in internal structure, hence the user will be using the same pattern, but providing different placeholder implementations.

Property 3 - Separation of function and function caller: In classical iteration algorithms, function evaluation is done inside the algorithm itself. Usually this is achieved by using some kind of expression language for representing the function as a string. These kinds of iterations are closed systems: they generate values, evaluate the function with those new values and check the condition of the iteration. This kind of closed system is not possible in CAE iterations. We do not have just a plain function, but a complete analysis represented as a workflow of business processess. Therefore the function is completely hidden and unknown to us. We will solve this by providing seperate placeholders for function value generation and function evaluation.

Property 4 - Pattern nesting: This refers to the possibility of one pattern reusing the other internally. Just like we can write a for-loop inside a while-loop, it should be possible for one CAE pattern to use another one inside its internal structure. We can refer to this as the level of the pattern.

Property 5 - Service oriented: The placeholders inside a pattern allow for customizability of the pattern. Of course, customizability can be achieved by allowing simple components to be plugged into the system such as jar files or dynamic libraries. However to gain the

benefits of collaboration and network-based programming we view our placeholders as entry points for web services thus applying to the service oriented architecture.

4.2.2 Pattern data

Several iteration data members will be present in the iteration. This data can either be exposed to or hidden from the user. Data that is exposed to the user should have clear semantics for example written in xsd schema [TBMM04] [BM04].
Two types of data will be explicitely exposed, that is the definition of an *input* and *output*. They can be represented in the xsd schema language as follows:

Listing 4.1: Input variable xsd schema definition

```
1  <complexType name="Input">
2      <sequence>
3          <element name="Nominal" type="double"></
               element>
4          <element name="Low" type="double"></element>
5          <element name="High" type="double"></element>
6      </sequence>
7  </complexType>
```

Listing 4.2: Output variable xsd schema definition

```
1  <complexType name="Output">
2      <sequence>
3          <element name="Value" type="double"></element
               >
4      </sequence>
5  </complexType>
```

Using type *Input* and *Output*, we can create our analysis workflow. We assume for our patterns the existence of the possibility to create such workflow with connected inputs and outputs.

4.3 Pattern 1: Analysis pattern

The analysis pattern allows us to model and execute some analysis workflow. The pattern assumes the existence of activity elements connected together

that represent this workflow. The pattern expects a number of variables of type *Input* at the start of the analysis together with a number of variables of type *Output* at the end of the analysis. Example implementations of the analysis pattern include simple formulas such as calculating the square of a value, but might just as well represent an entire workflow as the one from the Pennsylvania bridge case.

4.3.1 Structure

Figure 4.1 shows an ActiveVOS [End09] modeled representation of the analysis pattern having three inputs and two outputs. We do not make a difference between external or internal structure since this pattern executes only once.

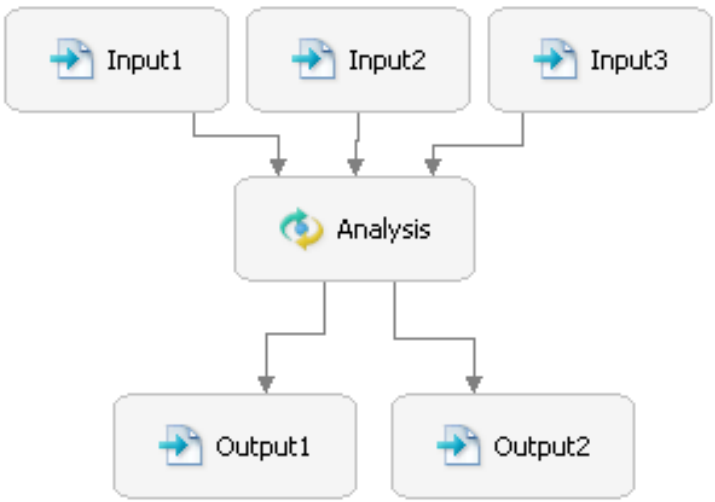


Figure 4.1: Structural representation of the analysis pattern.

4.3.2 Pattern nesting

Since this pattern contains no looping constructs and does not use any other patterns, we assign this pattern level 1.

4.3.3 Pattern data

The list below summarizes the data usage within the analysis pattern. For each member we provide a name, type, occurence and description.

Input variable:

Name: InputVar

Type: Type = Input

Occurences: Unlimited

Description: This input data is defined in the analysis pattern, but is globally available. All paterns can at any time access the information about *Inputs*.

Output variable:

Name: OutputVar

Type: Type = Output

Occurences: Unlimited

Description: As for the *Input* data, outputs are first defined in the analysis pattern, but are exposed globally.

Pattern input:

Name: PatternInput

Type: List of double

Size: Number of Input variables

Occurences: 1

Description: The analysis pattern expects as input a list of double values with the same size as the number of defined *Input* variables. This member is used as input to the analysis placeholder.

Analysis web service input:

Name: AnalysisInput

Type: List of double

Size: Number of Input variables

Occurences: 1

Description: The external web service expects as input a list of double values with exactly the same contract as the pattern input variable.

Analysis web service output:

Name: AnalysisOutput

Type: List of double

Size: Number of Output variables

Occurences: 1

Description: The external web service responds with a list of double with the same size as the number of defined *output* variables. This list is copied into the pattern output variable.

Pattern output:

Name: PatternOutput

Type: List of double

Size: Number of Output variables

Occurences: 1

Description: The analysis pattern responds to its caller with a list of double values with the same contract as the analysis placeholder.

4.3.4 Analysis placeholder

We can have an unlimited number of placeholders that together represent the analysis workflow. Figure 4.1 shows just one activity or web service called analysis, but in reality this can be a complete workflow of connected web services. The input to the analysis workflow should be list of Inputs equal to the number of connected input variables. The output of the web service should be a list of outputs equal to the number of connected output variables.

4.3.5 Execution

The input to the pattern is a list of Input values equal to the number of Input variables in this pattern. Using a BPEL copy statement, we will assign these incoming pattern values to the pattern Input variables to make those values globally available. Secondly we use BPEL's copy action again to put the assigned Input variable values to the web service input contract. For example the BPEL code below assigns the first pattern input value into the first input variable and then copies the input variable value into the first list item of the analyis web service input contract.

Listing 4.3: BPEL copy of pattern input to analysis input

```
<bpel:copy>
    <bpel:from>$PatternInput/ArrayOfDouble[1]</bpel:from>
    <bpel:to variable="Input1">
          <bpel:query>types1:Nominal</bpel:query>
    </bpel:to>
</bpel:copy>
<bpel:copy>
    <bpel:from variable="Input1">
          <bpel:query>types1:Nominal</bpel:query>
    </bpel:from>
    <bpel:to variable="AnalysisInput">
          <bpel:query>ns1:inputValues/double[1]</
             bpel:query>
    </bpel:to>
</bpel:copy>
```

These assignments actions will be repeated for the length of the pattern input list.
Next we invoke the actual analysis, which could be a workflow, but for the sake of simplification, that workflow is replaced by one external web service.

Using BPEL we will invoke the web service through the following invoke action:

Listing 4.4: Invoking the analysis web service

```
<bpel:invoke inputVariable="AnalysisInput" name="
   WebService_Name"
          operation="Some_Operation" outputVariable="
             AnalysisOutput"
          partnerLink="External_Provider" />
```

We retreive into *AnalysisOutput* a list of values that now need to be assigned to the iteration Output variables. Again this is achieved using BPEL copy statements. The pattern ends it execution by responding to the outside world with a list containing the resulting analysis output values.

Listing 4.5: BPEL copy of analysis output to pattern output

```
<bpel:copy>
      <bpel:from>$AnalysisOutput/ArrayOfDouble[1]</
         bpel:from>
      <bpel:to variable="Output1">
            <bpel:query>types1:Value</bpel:query>
      </bpel:to>
</bpel:copy>
<bpel:copy>
      <bpel:from variable="Output1">
            <bpel:query>types1:Value</bpel:query>
      </bpel:from>
      <bpel:to>$PatternOutput/ArrayOfDouble[1]</bpel:to>
</bpel:copy>
```

4.4 Pattern 2: Experiment loop pattern

This pattern is capable of enumerating, given a matrix of input values, over the matrix with a for-each loop construct, feeding one by one the next row of the matrix to the analysis workflow, collecting all resulting values into a summary.

4.4.1 Pattern structure

Using ActiveVOS, we can model the structure of this pattern as in figure 4.2.

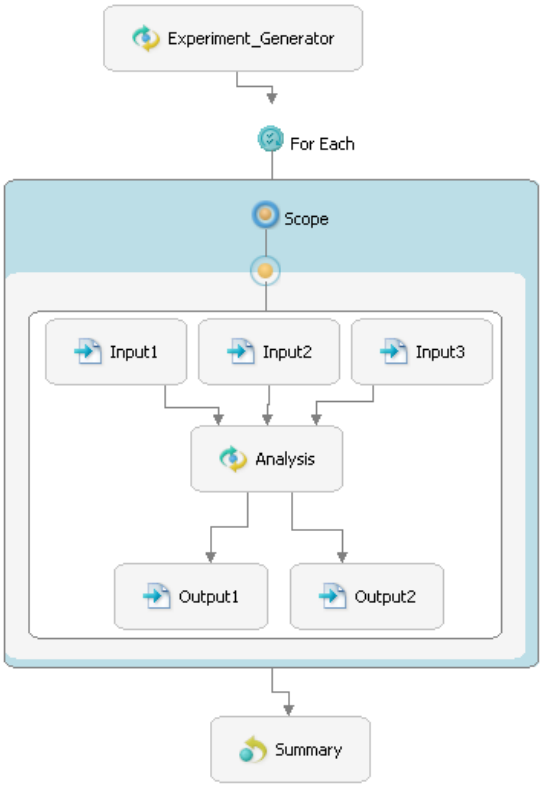


Figure 4.2: Structural representation of the experiment loop pattern.

The internal structure is one instance of the analysis pattern, whereas the external structure contains the experiment generator placeholder and a for-each loop construct to iterate a pre-determined times over the analysis pattern collecting all resulting analysis pattern output lists into a summary variable.

4.4.2 Pattern nesting

Level 2 since it composes a second pattern, the analysis pattern, in its internal structure. The pattern behaves as a container for the analysis pattern.

4.4.3 Placeholders

The experiment generator placeholder: this placeholder is responsible for generating a matrix of input values that will be handled by the for-each container.

The analysis pattern: this placeholder expects an instance of the analysis pattern reponsible for evaluating one row of input values.

Pattern data

The list below summarizes the data usage within the experiment loop pattern. For each member we provide a name, type, occurence and description.

Pattern input: None.

Experiment generator input: None.

Experiment generator output:

Name: GeneratorOutput

Type: Matrix of Double

Column Size: Number of Input variables

Row Size: Undetermined

Occurences: 1

Description: The experiment generator placeholder responds with a matrix of double values that should get fed to the internal for-each structure of the pattern.

Experiment counter:

Name: ExperimentCounter

Type: Integer

Occurences: 1

Description: Simple counter that gets increased at each step within the for-each construct.

Pattern output:

Name: Summary

Type: Matrix of Double

Column Size: Number of Input + Output variables

Row Size: ExperimentCounter

Occurences: 1

Description: Into this summary variable, we collect all results retreived from the analysis pattern.

4.4.4 Execution

The pattern does not expect any input. It will directly invoke the experiment generator web service. When the web service has finished, the resulting matrix of input values will be present in the experiment generator output variable. Next, the for-each loop will start which can be modelled in BPEL using a for-each statement. A BPEL for-each expression uses an internal counter (corresponding to our experiment counter) and needs to know the start counter value and maximum counter value. We will always start from 1 and loop until the counter has reached the length of the experiment generator output variable.

Listing 4.6: BPEL for-each construct

```
<bpel:forEach counterName="ExperimentCounter" parallel="
   no">
     <bpel:startCounterValue>1</bpel:startCounterValue>
     <bpel:finalCounterValue>count( $GeneratorOutput )</
        bpel:finalCounterValue>
       <bpel:scope>
       ...
       </bpel:scope>
</bpel:forEach>
```

In the BPEL scope statement we will execute the analysis pattern. One way of doing this is by viewing the analysis pattern as a web service. This way we can assign all pattern values and invoke accordingly. We can use the currenct experiment counter to retreive the next set of input values from the *GeneratorOutput* variable.

Listing 4.7: BPEL copy from experiment generator output to analysis pattern input

```
<bpel:copy>
     <bpel:from>$GeneratorOutput[$ExperimentCounter]/
        ArrayOfDouble[1]</bpel:from>
     <bpel:to>$AnalysisPatternInput/ArrayOfDouble[1]</
        bpel:to>
</bpel:copy>
```

The analysis pattern as we know will answer with a variable *PatternOutput*, which is a list of doubles. We collect all these resulting analysis lists into one big matrix, which will yield the output of the experiment loop pattern.

Listing 4.8: BPEL copy from analysis output to experiment loop summary

```
<bpel:copy>
     <bpel:from>\$AnalysisPatternOutput</bpel:from>
     <bpel:to>\$Summary[\$ExperimentCounter]</bpel:to>
</bpel:copy>
```

4.4.5 Random experiment loop pattern

We can model Monte-Carlo methods, DOE's and Table methods with an experiment loop pattern. However for those methods that generate random numbers, that is Monte-Carlo methods and Random DOE's, we need to make a slight adaptation in the data usage of the experiment loop pattern. We need to proivde the number of row values that should be generated by those random methods. This information is unneeded for non-random methods because the number of experiments is generated based on the number of inputs and the generator algorithm itself (see Section 3.2.2).
This pattern inherits from the normal experiment pattern and adapts the data usage as follows:

Pattern input:

Name: NumberOfExperiments

Type: Integer

Occurences: 1

Description: We ask as input to the pattern a number of experiments that should get generated by the experiment generator placeholder.

Experiment generator input:

Name: GeneratorInput

Type: Integer

Occurences: 1

Description: The experiment generator placeholder input expects a number of experiments. This information is retreived from the *NumberOfExperiments* variable in the pattern input.

4.4.6 Iteration mapping

Table methods, Design of experiments and Monte-Carlo methods all generate their input values in a deterministic way. We state that these iterations can thus be modelled using the experiment loop pattern. We implement this input generation using the experiment generator placeholder. Since this placeholder is in the external structure of the pattern, thus outside of and prior to the internal loop, determinism is preserved.

Within these iterations we further differentiate Table methods and Non-Random Design of Experiments from Random Design of Experiments and Monte-Carlo methods. Table methods and Non-Random DOE's designs can be modelled using the classical experiment loop pattern. Random DOE's and Monte-Carlo methods should be modelled using the manual experiment loop pattern.

4.5 Pattern 3: Optimization loop pattern

The optimization loop pattern is capable of modeling all optimization algorithms by providing a suitable algorithm that fits into the pattern and providing a stop condition to the pattern. The pattern will repeatedly execute the experiment loop pattern until the stop condition is satisfied. We call one execution of the experiment loop pattern, an iteration. All iteration matrices are collected in a resulting summary.

4.5.1 Problem description

Providing a pattern for optimization is a lot more challenging compared to modelling DOE's or Monte-Carlo methods. We can devide the complexity into two categories. The first one is related to the complex internal structure of optimization algorithms in general, whereas the second problem is related to a specific subcategory of local optimizations.

Problem 1: Complex internal structure

This problem is related to the fact that we can no longer ask for just one set of input values to be executed because the next set is depending on the outcome of the previous set. In other words, we need to provide information from the previously executed values to the algorithm before we can start the next iteration. This means that our pattern will be responsible for providing to the optimization algorithm as input the set of previously calculated values.

Problem 2: Local optimization algorithms

The problem is related to property 3 in Section 4.2.1 where we explained how we are forced to have a seperation in function and function caller. For local

optimization algorithms (See Section 3.3.1), this problem goes even further due to its need for gradient information. Most local optimizations need to know the derivative of the original function in order to succesfully climb up or down the problem space. However, since the optimization algorithm does not and should not know anything about the original function, it cannot know the derivative function. So then, how can the algorithm decide what should be the next set of values, how can it decide in what direction to search? Luckily in the domain of numerical analysis, this can solved by the application of the **finite difference approximation method**.
The mathematical trick directly follows from the definition of a derivative in a point x:

$$f^{'}(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}. \tag{4.1}$$

Instead of taking x and filling it into $f^{'}(x)$, we take a very pre-determined small fixed value h and fill it into the right-hand side of equation 4.1. This way we approximate the actual derivative value and avoid having to know about the derivative function.
Note how global optimization alorithms (See Section 3.3.1) will not face this problem because they do not need use any gradient information. Instead they provide a set of randomized values based on the result of the previous set.

4.5.2 Pattern structure

Figure 4.3 shows the structural representation of the optimization loop pattern. The external structure of the pattern consists of a while loop that runs until one of the two following conditions hold: the maximum number of iterations is reached or the optimization has converged. The latter is specified through our first placeholder called isConverged. The internal structure consists of an if statement with two branches: the first branch executes only once that is the very first loop (or more specifically when the iteration counter is equal to 1), the second branch executes all other times. Both branches contain a placeholder for the experiment loop pattern (see Section 4.4).

4.5.3 Pattern nesting

We assign level 3 to this pattern because it contains in its internal structure a pattern (the experiment loop pattern) of level 2.

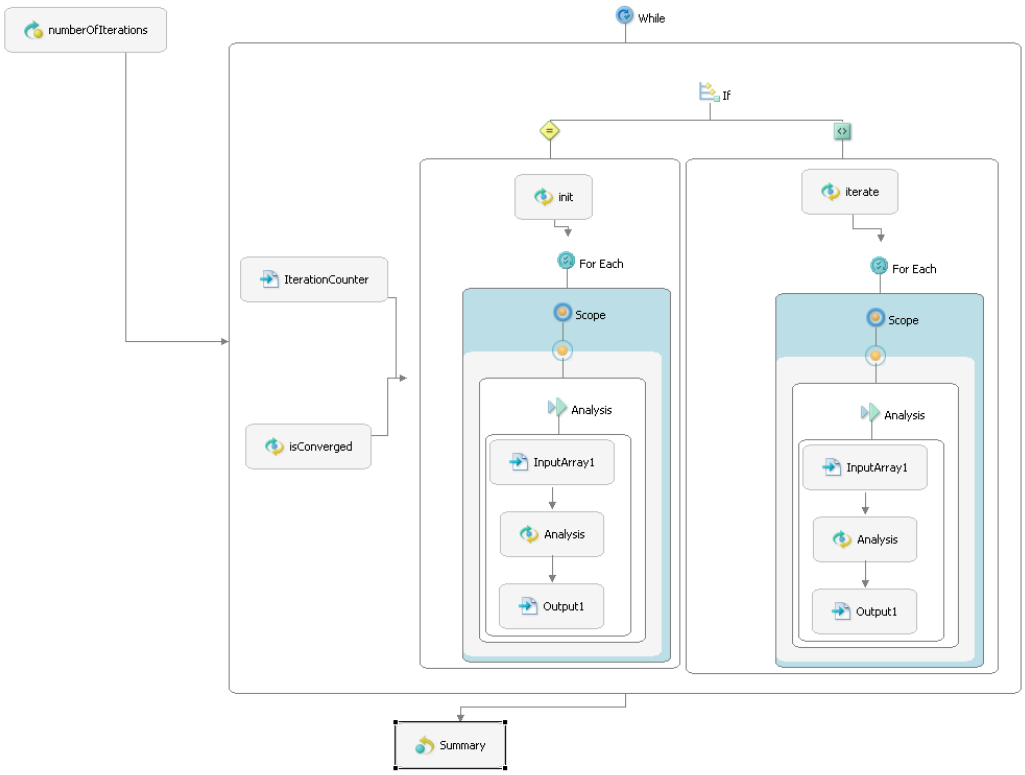


Figure 4.3: Structural representation of the optimization loop pattern.

4.5.4 Placeholders

IsConverged placeholder: External web service that should return a simple boolean value. When isConverged evaluates to true, the pattern will end the main iteration loop.

Experiment loop pattern placeholder 1: We provide an instance for the pattern. As we know from the pattern, instantiating means providing a placholder for the experiment generator and providing some analysis. Our first instance in the first branch of the if structure, will implement the placeholder with an init web service. This service will allow to initialize the optimization algorithm, expecting a first set of values to be executed by the experiment loop.

Experiment loop pattern placeholder 2: The second instance in the second branch of the if structure, will implement its placeholder with the actual optimization specific logic. This can be any implementation leading from local, global to multi-objective algorithms. But whatever algorithm provided, they all provide us the next set of input values that will be passed to its experiment loop pattern. Note how the first time this placholder runs, it will use as input the resulting values from the first experiment loop in the first branch of the if structure.

4.5.5 Pattern data

The list below summarize the data usage within the optimization loop pattern. For each member we provide a name, type, occurence and description.

Pattern input:

Name: NumberOfIterations

Type: Integer

Occurences: 1

Description: Entry point for the pattern, that tells the while structure the maximum number of times it should run. The pattern can run less than this maximum number when the isConverged placeholder tells the optimization algorithm it has converged.

While loop counter:

Name: IterationCounter

Type: Integer

Occurences: 1

Description: This variable keeps track of the number of iterations performed so far. It is used by the if structure to decide which branch to execute and by the while structure to check if the maximum number of iterations is not yet reached.

IsConverged placholder input: None

IsConverged placeholder output:

Name: IsConvergedOutput

Type: Boolean

Occurences: 1

Description: Contains the result of the invocation of the external IsConverged web service. When the value results true, the while structure will ends its loop.

Init placeholder input:

Name: InitInput

Type: List of Double

Size: Number of Input variables

Occurences: 1

Description: One of the things the algorithm likes to know is where in the problem domain he should start the search for the optimal value. Therefore, we pass as input to the init web service, a list of start values, one for each input variable. By default, we can take the Nominal value specified in the Input variable definition (see Section 4.2.2).

Init placeholder output:

Name: InitOutput

Type: Matrix of Double

Column Size: Number of Input variables

Row Size: Undetermined

Occurences: 1

Description: As a result from the external Init web service, we retreive a first set of values to be executed by its experiment loop pattern. For example for a local optimization algorithm, this resulting matrix will contain in its first row the input values specified in *InitInput* whereas the next rows contain the finite differenced values.

Iterate placeholder input:

Name: IterateInput

Type: Matrix of Double

Column Size: Number of Output variables

Row Size: Number of rows retreived from the *Summary* variable from the experiment loop pattern.

Occurences: 1

Description: We pass to the actual optimization engine (again an external web service), the values calculated by the experiment loop pattern. Recall from Section 4.4.3 that the experiment loop pattern collects all its calculated values into one big summary. We use this information to pass as input to Iterate placeholder, so that the optimization engine can decide based on this information where to go to next in his problem domain, hence gradually converting to the optimal value.

Iterate placeholder output:

Name: IterateOutput

Type: Matrix of Double

Column Size: Number of Input variables

Row Size: Undetermined

Occurences: 1

Description: Exactly the same structure as the *InitOutput* variable. Indeed, it provides the same kind of information. It gives us the next set of input values to be executed by the experiment loop pattern.

Pattern output:

Name: Summary

Type: Matrix of Double

Column Size: Number of Input + Output variables

Row Size: Sum of all $ExperimentCounter_i$ where is the iteration counter.

Occurences: 1

Description: This variable collects all summaries from all experiment loop pattern runs together.

4.5.6 Execution

The pattern expects in its input a maximum number of times the while structure should execute its internal structure. A current iteration counter will be held into the *IterationCounter* variable. The while structure itself can be modeled using a BPEL while construct. The required field for a BPEL while construct is a BPEL condidtion. The BPEL code listing below shows how it is written for our pattern.

Listing 4.9: BPEL while construct and invocation of the IsConverged web service

```
1  <bpel:while>
2  <bpel:condition>$IterationCounter &lt; $
       NumberOfIterations and not( $IsConvergedOutput = "true
       " )</bpel:condition>
3  <bpel:flow>
4      <bpel:invoke inputVariable="IsConvergedInput" name="
           IsConverged" operation="hasConverged"
5                           outputVariable="
                               IsConvergedOutput"
6                           partnerLink="Some_Provider">
7      </bpel:invoke>
8      ...
9  </bpel:flow>
10 </bpel:while>
```

Line number 2 shows us the XPath [CD99] expression for the header of the while construct and line 4 shows the invocation of the external IsConverged web service. The latter will be executed as a first step within the while body. Next we model our if structure with a BPEL if construct where the first branch executes only when the iteration counter is 1 as seen in the piece of code below:

Listing 4.10: BPEL if-else construct

```
1  <bpel:if>
2     <bpel:condition>$IterationCounter = 1</bpel:condition>
3     <bpel:flow>
4        ...
5     </bpel:flow>
6     <bpel:else>
7         <bpel:flow>
8            ...
9         </bpel:flow>
10    </bpel:else>
11 </bpel:if>
```

If we are in the if branch, we invoke the init service with a set of start values equal to the number of inputs and retreive a matrix of values. From here we execute the experiment loop pattern which is described in Section 4.4.4, hence we do not repeat its code. If we are in the else branch, we invoke the Iterate placeholder and pass as information the matrix of previously

calculated values. Then again executing the experiment loop pattern with the result of the Iterate web service.

Listing 4.11: BPEL copy from experiment loop output to iterate web service input

```
...
<bpel:else>
    <bpel:copy>
           <bpel:from variable="ExperimentLoopOutput" />
           <bpel:to variable="IterateInput" />
    </bpel:copy>
    <bpel:invoke inputVariable="IterateInput" name="
       Iterate" operation="iterate"
           outputVariable="IterateOutput"
            partnerLink="Provider1">
    </bpel:invoke>
    <bpel:copy>
           <bpel:from variable="IterateOutput" />
           <bpel:to variable="ExperimentLoopInput" />
    </bpel:copy>
    <bpel:forEach counterName="ExperimentCounter"
       parallel="no">
    ...
    </bpel:forEach>
</bpel:else>
...
```

At the end of each iteration we collect the summary from the experiment loop pattern and put into our collective summary for the optimization loop pattern as in the code below:

Listing 4.12: BPEL copy from experiment loop output to optimization loop summary

```
<bpel:copy>
     <bpel:from>$ExperimentLoopOutput</bpel:from>
     <bpel:to>$Summary[$IterationCounter]</bpel:to>
</bpel:copy>
```

4.5.7 Iteration mapping

In our research we have studied the internal structure of optimization algorithms. This structure can be decomposed into four structures: a while-loop

construct, a convergence check, an initialization step and the generation of input values. These four constructs are seperate entities in our pattern. Therefore we state that all optimizations following the same structure can be modelled using the optimization loop pattern. The challenge occurs in the generation of input values.
First of all this generation is non-deterministic. This means we should place our generator placeholder in the pattern's internal structure. The placeholder is then integrated in the while loop and previous function values can be passed as input information to the placeholder.
Secondly, an optimization should be capable of generating a set of input values without knowledge of the original function or derivative function. Global optimizatons such as evolutionary algorithms meet these requirements because they evaluate the fitness based on global function information only. Local optimizations will need adaptation if they are tightly coupled with the derivative function. They need to change their gradient calculation from an analyis-based gradient method to a finite difference approximation method. We will show how this can be done in Section 5.4.

Chapter 5

Proof of concept implementations

5.1 Introduction

In this chapter we will provide implementations for the proposed patterns to proof that the concept is correct and usable. In Section 5.3 we will implement the experiment loop pattern and in Section 5.4, we will make an implementation for the Optimization loop pattern. Various programming languages and frameworks are used to make these implementations ranging from Java [GJSB00] and Axis Web Services [PR09] to .NET [Pla03] and ASP.NET [Esp08].

5.2 Analysis pattern:

5.2.1 Example 1: Squared function

Since this is our first pattern, let us not make it too complex. We will use one *Input* variable and one *Output* variable. Let us simply take the squared function: $f(x) = x^2$. Note how the function correctly maps to the number of Input variables in our worklow. We will implement the analysis placeholder as an external web service. Different technologies exist out there and for this example we have used Java(1.6) together with Axis Web Services(1.4) to implement and generate the web service. The squared function is written in Java as in the code below. We expect one double value in its input argument to map on *AnalysisInput* and one double value to map on *AnalysisOutput*.

Listing 5.1: Squared function implementation

```
public double squareValue(double value) {
        return value * value;
    }
```

Using Axis Web Services, we can then generate the description WSDL [CCMW01] file that provides us the contract for the web service. This WSDL is the binding between our pattern and the external web service. The generated WSDL file is listed in Appendix C.1.1. From within our analysis pattern we can then invoke the squared function as follows:

Listing 5.2: Invoking the squared function web service

```
<bpel:invoke inputVariable="squareValueRequest" name="
   FormulaService"
          operation="squareValue" outputVariable="
             squareValueResponse"
          partnerLink="Provider" />
```

We can visualize this analysis implementation through figure 5.1.

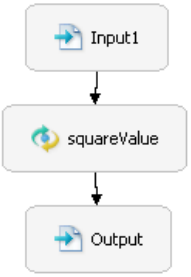


Figure 5.1: Structural representation of the Squared Function analysis pattern implementation

5.2.2 Example 2: Rosenbrock function

Now for a more challenging analysis pattern implementation. We will implement the Rosenbrock function, a non-convex function that is typically used for testing optimization algorithms due to its complex problem space. The reason is that the global minimum is located inside a long narrow parabolic shaped flat valley, making the convergence a lot more difficult (see figure 5.2).

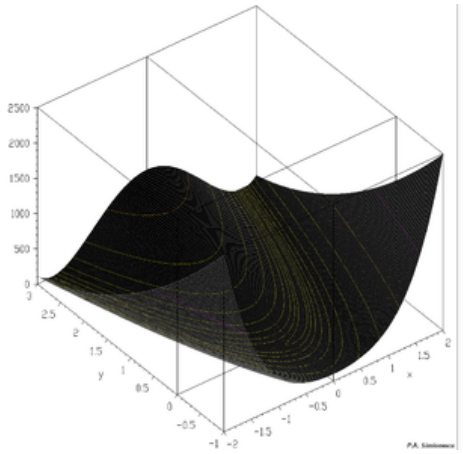


Figure 5.2: Plot of the Rosenbrock function of two variables

The generalized version of the Rosenbrock function is defined by:

$$f(x) = \sum_{i=1}^{N-1} [(1 - x_i)^2 + 100(x_{i+1} - x_i^2)^2] \tag{5.1}$$

This analysis pattern implementation will be useful later when we implement the optimization loop pattern in Section 5.4. We will use two *Input* variables and one *Output* variable. This time we have used .NET and ASP.NET to write the web services for no other reason than to try out different technologies since that is the beauty of web services: they are language independent. The piece of code below shows the .NET implementation of the external web service for the Rosenbrock function:

Listing 5.3: Rosenbrock function implementation

```
1     [WebMethod]
2     /*
3      * Multi-dimensional Rosenbrock formula
4      * Expecting as input at least 2 inputValues
5     */
6     public double Rosenbrock(double [] inputValues) {
7         double x = 0, y = 0;
8         double result = 0;
9         for (int i = 0; i < inputValues.Length - 1; i++)
10        {
11            x = inputValues[i];
12            y = inputValues[i + 1];
```

```
13              result += Math.Pow((1 - x), 2) + (100 * Math.
                    Pow((y - Math.Pow(x, 2)), 2));
14          }
15          return result;
16      }
```

As you can see we generalize over the number of inputs mapping on the *AnalyisInput* variable and return one double value mapping on the *AnalysisOutput* variable. Using ASP.NET we can automatically create the WSDL file as shown in Appendix C.1.2.
Using this WSDL file we can invoke the web service from within the pattern as follows:

Listing 5.4: Invoking the Rosenbrock function web service

```
1 <bpel:invoke inputVariable="Rosenbrock" name="Rosenbrock"
2           operation="Rosenbrock" outputVariable="
                RosenbrockResponse"
3           partnerLink="Provider" />
```

Finally we can visualize the implementation using ActiveVOS as shown in figure 5.3.

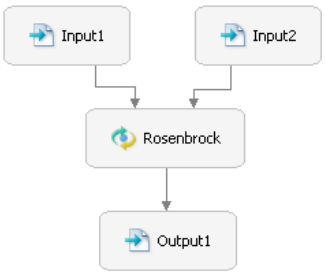


Figure 5.3: Structural representation of the Rosenbrock Function analysis pattern implementation

5.3 Experiment loop pattern

Let us implement the random experiment loop pattern in order to model a Random Design of Experiments method. In order to make a pattern instance

we need to specify implementations for its placeholders. For the manual experiment loop pattern this means implementing one analysis pattern and one experimeng generator placeholder.

5.3.1 The analysis placeholder

For this pattern we will simply reuse the squared function pattern implementation we have proposed in Section 5.2.1. Recall that this implementation has one Input and one Output. For the Input we will assign a low value of 0 and a high value of 100.

5.3.2 The experiment generator placeholder

We will provide a simple Random DOE Method generator written in Java and exposed using Axis. The contract of this web service is directly given by the pattern. As input we look at the *GeneratorInput* which is an Integer and as Output we expect (according to *GeneratorOutput*) a matrix of Double. This leads to a possible implementation as given in the code below:

Listing 5.5: Random generator web service implementation

```
1  public double[][] generateDOE(int num) {
2          double [][] values = new double[inputs.size()][];
3          Random generator = new Random();
4          for(int i = 0; i < inputs.size(); i++) {
5              values[i] = new double[num];
6              Input input = inputs.get(i);
7              for(int j = 0; j < num; j++) {
8                  double value = (double)generator.nextInt
                       ((int)input.high) + generator.
                       nextDouble() + input.low;
9                  values[i][j] = value;
10             }
11         }
12         return values;
13 }
```

The code generates Random double numbers in the range between the given low and high of the input. In our case, the resulting matrix of Doubles will have only one column since we have only one Input defined in the analysis

pattern. Using Axis this code generates the WSDL file as given in Appendix C.2. Using this binding we can invoke from within the pattern as follows:

Listing 5.6: Invoking the random generator web service

```
1 <bpel:invoke inputVariable="generateDOERequest" name="
     generateDOE"
2           operation="generateDOE" outputVariable="
              generateDOEResponse"
3           partnerLink="Provider" />
```

5.3.3 Execution

This ends the implementation for the random experiment generator pattern and allows the complete BPEL process to be run as a web service. Figure 5.4 shows the starting execution point of our implementation. In this exam-

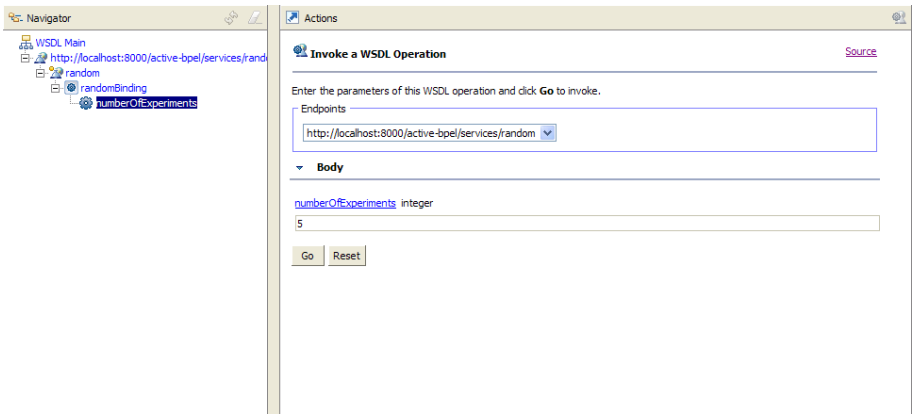


Figure 5.4: Random experiment loop pattern execution

ple we ask to generate 5 experiments. When pressing the "go" button the pattern will execute five times our squared function and collect all results in a summary. Using ActiveVOS we can get a global execution overview and watch the final summary. This is shown in figure 5.5. Notice indeed 5 input random numbers between 0 and 100 and their corresponding squared values.

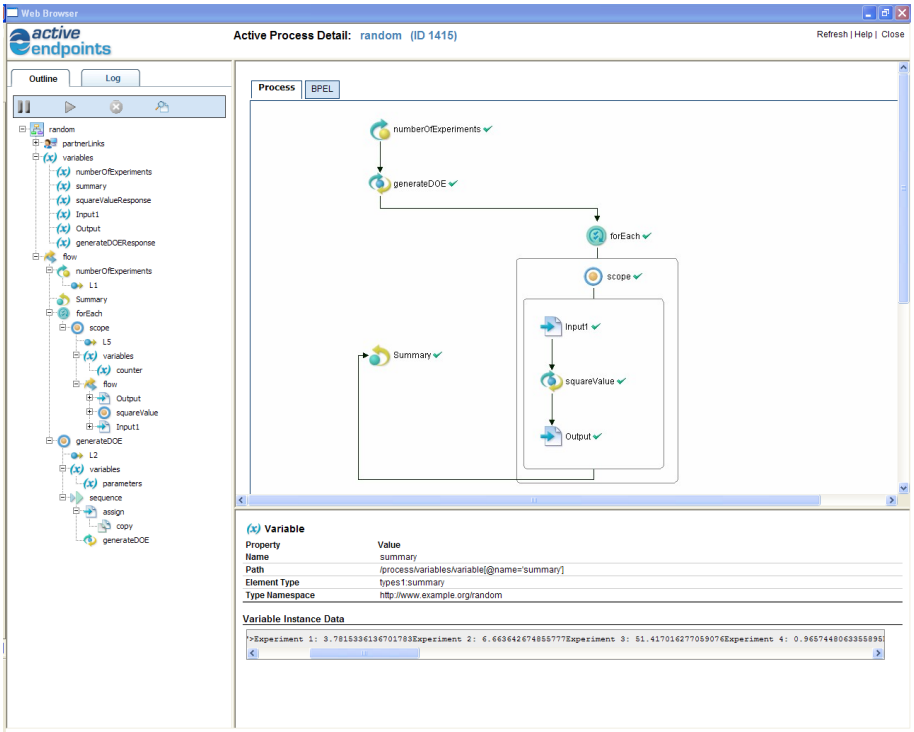


Figure 5.5: ActiveVOS monitoring BPEL process

5.4 Optimization loop pattern

We will implement the optimization loop pattern with an L-BFGS (Limited-memory Broyden Fletcher Goldfarb-Shanno) optimization algorithm [Boc00] written in .NET C#. This algorithm is an optimization algorithm belonging to the category of local optimizations using gradient information to search for the optimal value. This time we need to provide implementations for 3 placeholders: We should provide the convergence criterium, implement the initialization experiment loop and provide the actual algorithm core in the second experiment loop pattern. Remember that using these placeholders we are able to model any possible optimization algorithm. Let us explain this in more detail using the following code listing that demonstrates the usage of the L-BFGS algorithm:

Listing 5.7: L-BFGS Usage

```
1 static void Main(string[] args)
2 {
3     lbfgs.lbfgsstate state = new lbfgs.lbfgsstate();
```

```
4       double [] s = new double[0 + 2];
5       int n = 2;
6       int m = 2;
7       s[0] = 20;
8       s[1] = 30;
9       lbfgs.minlbfgs(n, m, ref s, 0.0, 0.0, 0.0, 0, 0, ref
            state);
10      while (lbfgs.minlbfgsiteration(ref state))
11      {
12          x = state.x[0];
13          y = state.x[1];
14          state.f = Math.Pow((1 - x), 2) + (100 * Math.Pow
                ((y - Math.Pow(x, 2)), 2));
15          state.g[0] = -2 + (2 * x) - (400 * x * (y - Math.
                Pow(x, 2)));
16          state.g[1] = 200 * (y - Math.Pow(x, 2));
17      }
18  }
```

Line 3-9 intializes the algorithm. Consequently we will use these lines in the first experiment loop placeholder of the pattern. Line 10 shows the while loop, which will no longer occur in the pattern implementation since this is the task of the pattern itself. Furthermore, line 10 performs an actual iteration in the algorithm and returns whether or not the algorithm has converged. The statement performs two tasks at once, hence we should split up these tasks into two seperate responsibilities. The convergence check should be moved to the IsConverged placeholder and the Iteration should be moved to the second experiment loop placeholder. This structure is always present in any algorithm as it defines what an Optimization algorithm is supposed to do.
The code will need further refactoring as line 14, 15 and 16 require knowledge about the original function. Line 14 shows the Rosenbrock function, which we will model separately in the Analysis placeholders of the Experiment loop patterns. Line 15 and 16 use the Rosenbrock's derivative function. As explained in Section 4.5.1, we are unable to use the derivative function and should replace it with the finite difference approximation method.

5.4.1 Experiment loop pattern 1 placeholder

This pattern needs 2 internal placeholders. One for the initialization of the algorithm resulting in the execution of a first set of values and one placeholder

for the internal analayis. As analysis we will choose the Rosenbrock function as introduced in Section 5.2.2. We will use two *Input* variables and one *Output*. For the experiment generator placeholder we will write an external web service that allows us to initialize the optimization algorithm and as a result gives a first set of values to be executed by the experiment loop pattern. The code listing 5.8 shows how we wrote the init web service:

Listing 5.8: Init web service implementation

```
1   static lbfgs.lbfgsstate state;
2   static double[] s = new double[0];
3   static int n = 0;
4   static int m = 0;
5
6   [WebMethod]
7   public double[][] init(int [] startValues)
8   {
9       state = new lbfgs.lbfgsstate();
10      isConverged = false;
11      n = startValues.Length;
12      m = n;
13      s = new double[n];
14      for (int i = 0; i < startValues.Length; i++)
15      {
16          s[i] = startValues[i];
17      }
18      lbfgs.minlbfgs(n, m, ref s, 0.0, 0.0, 0.0, 0, 0,
            ref state);
19
20      double [][] firstValues = new double[startValues.
            Length][];
21      for (int i = 0; i < startValues.Length; i++)
22      {
23          firstValues[i] = new double[1];
24          firstValues[i][0] = startValues[i];
25      }
26      return firstValues;
27  }
```

We do exactly the same as in code list 5.7, that is, we just initialize the algorithm. It differs in that we now should return a first set of values back to the pattern. In this case we just return the transposed of the startValues parameter.

5.4.2 Experiment loop pattern 2 placeholder

We should now provide an implementation for the second experiment placeholder. This placeholder is responsible for executing the actual core of the algorithm. We have the added complexity of changing the code from 5.7 to use the finite difference method for calculating the gradients. Code listing 5.9 below shows our implementation.

Listing 5.9: Iterate web service implementation

```
1   static Boolean isConverged;
2
3   [WebMethod]
4   /*
5    * Input are the computed function values with length
         n + 1
6    * Output are the new set of input values to be
       computed with dimension (n, n + 1)
7    */
8   public double[][] iterate(double [] functionValues)
9   {
10      state.f = functionValues[0];
11      for (int i = 1; i < n + 1; i++)
12      {
13          state.g[i - 1] = functionValues[i];
14      }
15
16      if (!lbfgs.minlbfgsiteration(ref state))
17      {
18          isConverged = true;
19          double [][]result = new double[n][];
20          for (int i = 0; i < result.Length; i++)
21          {
22              result[i] = new double[1];
23              result[i][0] = state.x[i];
24          }
25          return result;
26      }
27
28      double [][] newValues = new double[n][];
29      for (int i = 0; i < newValues.Length; i++)
30      {
31          newValues[i] = new double[n + 1];
32          newValues[i][0] = state.x[i];
```

```
33              for (int j = 1; j < newValues[i].Length; j++)
34              {
35                  if(i == j - 1)
36                      newValues[i][j] = state.x[i] + ((Math
                            .Abs(state.x[i]) + 0.001) * 0.001)
                            ;
37                  else
38                      newValues[i][j] = state.x[i];
39              }
40          }
41          return newValues;
42      }
```

First of all note from the input parameter that this method expects function values. Indeed every time we invoke this service we should pass the previously calculated results. The first time this service is invoked these results will come from the summary of the initialization experiment loop. Line 10 - 14 fill in these values into the function and gradient state of the algorithm. Line 16 then performs the actual core of the algorithm. In this case it will decide based upon the information passed into *state.f* and *state.g*, the next set of input values. These values will be placed into *state.x* by the algorithm. Note on line 18 that if the *minlbfgsiteration* function returns false, that the algorithm has converged. If this happens, we will update our *isConverged* boolean variable accordingly. On line 28 - 40, we fill into the first column of the *newValues* matrix, the new *state.x* values and add to the remaining columns the new gradient values calculated through the finite difference method. The *newValues* matrix gets returned to the pattern and the experiment loop will enumerate over these new values in order to calculate the next set of function values.

5.4.3 IsConverged placeholder

This placeholder is responsible for informing our optimization pattern with the information of convergence. Recall from code listing 5.9 line 18 that we use the *isConverged* Boolean variable to hold this information. We can now write our web service through the following piece of code:

Listing 5.10: IsConverged web service implementation

```
1     [WebMethod]
2     public Boolean hasConverged()
3     {
4         return isConverged;
5     }
```

The generated WSDL code for the different placeholders is attached in Appendix C.3. All placeholders are present in the same WSDL file as separate service operations.

5.4.4 Execution

Now that we have provided implementations for all placeholders, we can execute the pattern using ActiveVOS. Figure 5.6 shows the starting execution point. We pass 100 as maximum number of iterations and press the "go" button. Browsing trough the ActiveVOS monitor, we can watch the results

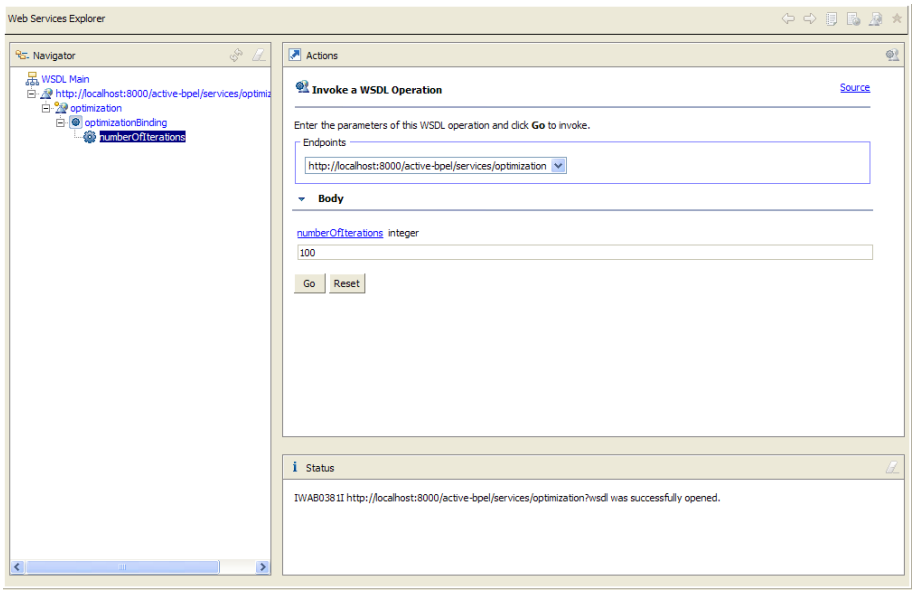


Figure 5.6: Optimization loop pattern execution

of the exucution as seen in figure 5.7. On the left hand side we have an overview of all variables and their final value at the end of the optimization. The *iterationCounter* shows a value of 42. This means that 42 iterations have

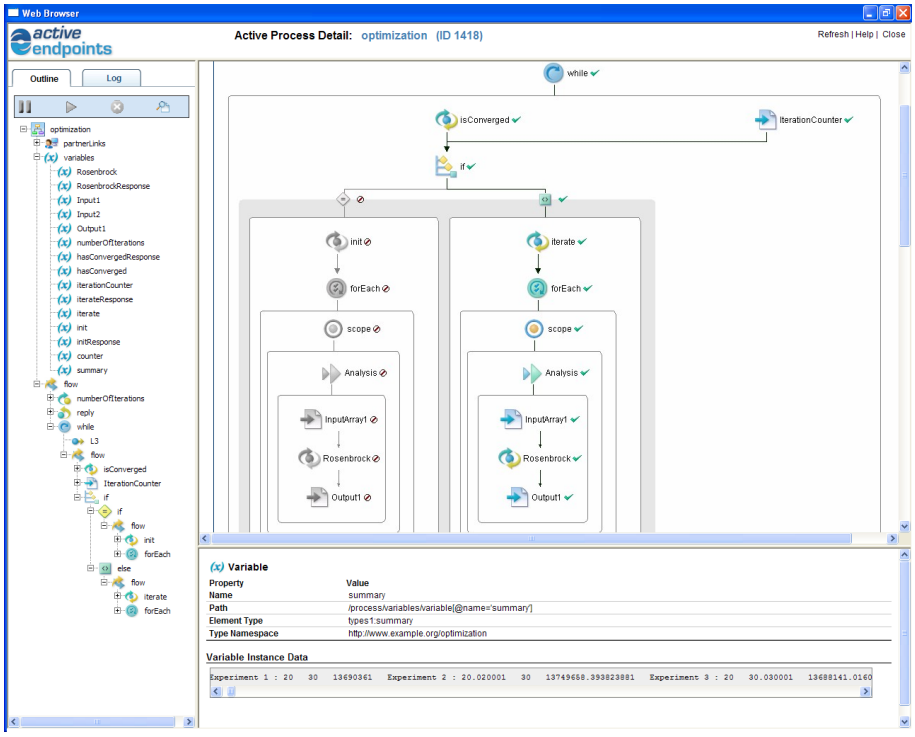


Figure 5.7: ActiveVOS monitoring BPEL process

occurred before the optimal value has been found. This optimal value is seen by clicking the *Output1* variable. It shows an optimal value of 7.51435. This means the optimization algorithm got stuck in a local optimum since the global optimal value for the Rosenbrock function is 0. As we know from the property of local optimizations this behavior is expected (see Section 3.3.1). The corresponding input values for this optimal value are seen in the *Input1* and *Input2* variables. They show us a value of 3.72340 and 13.89494 respectively. All this information can also be found in the *summary* variable, giving us a list of all executed experiments with corresponding input and output values.

Chapter 6

Future Work

We have proposed several patterns that allow to model the different iterations in the domain of CAE/MDO. We claim that a fast implementation of an iteration is possible thanks to the use of placeholders within a single pattern. However, the patterns sould still be designed in a way that one pattern covers as many iterations as possible. This balance between coverage and ease of implementation should be evaluated on each individual iteration. For example, we could fine-tune the optimization loop pattern in the context of multi-objective optimization algorithms as presented in Section 3.3.2. At the moment the optimization loop pattern proposes only one placeholder that checks for convergence. Due to this single placeholder, implementing a convergence check for multi-objective is more difficult. We can ease this work by providing in the optimization loop pattern separate convergence placeholders, one for each objective. We move the complexity from the internal algorithm to the pattern structure itself.

Second fine-tuning can happen on the level of data usage within the different patterns. We presented schemas for representing *Input* variables and *Output* variables. At the moment these schemas are somewhat limited. The reason is that these schemas are of secondary importance compared to the patterns itself. Still, it might be good to formalize these schemas in more detail. For example, data elements that should be added are those that are used within Robustness and Reliability iterations. These iterations require inputs to have information about their distribution behavior. This includes information about the sigma value, type of distribution (normal, lognormal, weibull, etc.) and information about the distribution's low and high bound.

Another possible extension to the system, is to allow patterns to be connected to each other. This way we can model a series of iterations. On top of that we could provide the option for the next iteration to start from the optimal set of values from the previous iteration. It will allow us to mimic the scenario described in Section 3.3.3, where we performed three consecutive optimizations with the final SQP optimization starting from the optimal value of the previous SAE optimization.

Yet another addition could be to allow Response Surface Models [MMAC08] to be incorporated in place of analysis placeholders. RSM's provide an approximation for the internal analysis. We could for each *Output* variable provide the option to be evaluated either by the internal analysis or by the internal RSM.

In Section 2.4 we have explained, for the first two requirements, the solutions offered by the conceptual workflow language. It would be benefical to show how these constructs can be used in our solution. We recall the reader that data ports were presented to model the advanced data usage within MDO applications. These data ports can be used in our solution to model the different data elements present in the proposed patterns. The sub-workflow construct was presented as a solution for modularizing croscutting concerns. Looking at the design for the optimization loop pattern as presented in Section 4.5, we can see that the analysis patterns present in the two experiment loop patterns are identical. We say that the analysis workflow is scattered accross the design. Future work can remedy this croscutting concern using the proposed sub-workflow construct.

A final continuation consists in providing more concrete example implementations as those presented in Chapter 5. This will convince users in the MDO domain that implementing their iterations is quick and easy. For example, in the context of Design Space Exploration, we have implemented a Random DOE. It would be helpful to provide an implementation for orthogonal DOE's as well. In the context of optimization, we have implemented a local-optimization algorithm. We could add an implementation for a global optimization method, such as an evolutionary algorithm.

We have stated several times that this thesis completes the conceptual framework. This means that all requirements defined in the requirements analysis are met and that in a next step we can start on an actual first language implementation. This implementation will happen in YAWL [Kne04] due to its formal approach and support by a large and active academic and research

group. But before this implementation can start, we have to translate our proposed patterns from BPEL into YAWL. This translation can happen as soon as the language extension newYAWL [Min07] is ready. newYAWL will include all the iteration constructs that are required by our patterns.

Chapter 7

Conclusion

Workflow languages can be used to model business processess. In theory they can in fact be used to model almost any process, yet in practice, we see that this statement does not hold. Some processess like engineering processess are not being modelled using existing workflow languagues. The reason is that the complexity of these processess pushes today's workflow languages to their limit. Recent research has studied this complexity. The outcome of this study was the definition of a set of problems and requirements that need to be met. The first problem acknowledges the use of many different concerns that get scattered accross the design. A second problem states the intensive data usage inside the process and a third problem recognizes the use of advanced iteration constructs. The first two problems have been solved by introducing a conceptual language built upon YAWL that leverages the language with sub-workflows and data ports.
This thesis provides a solution to the third and final problem. For that we took two steps: we studied the use of iterations within MDO applications and proposed a set of workflow patterns that would allow these iterations to be modelled. The iterations have been studied in Chapter 3. There, we present a classification of commonly used iterations within MDO applications. This is a classification as seen from the eyes of a structural engineer. The classification has three parts: Design Space Exploration (DOE) iterations, Optimization iterations and Robustness and Reliability iterations. We have taken both a theoretical and practical approach to fully understand these iterations.
In a next step in Chapter 4 we have proposed a set of workflow patterns. These patterns allow us to model the previously classified iterations. We reclassify the iterations through the eyes of a software engineer by mapping the iterations onto these patterns. A first pattern, which we labeled the Analysis Pattern, allows us to model the design behind the iteration. This

design can range from a simple formula to calculate the square of a number to a complex engineering study such as the accoustics within a car's engine. A second pattern, called the experiment loop pattern allows us to model iterations that are deterministic in their input generation. These are all DOE's and Monte-Carlo methods. A final pattern, the optimization loop pattern, allows us to model all optimization iterations. We showed that some of these optimizations will need adaptation to fit into this pattern. Local optimizations will have to change their gradient calculation from analytical to a finite difference method.

To prove our concept, we have selected in Chapter 5 for each pattern an iteration that we implemented into our pattern. We do this by implementing the required placeholders and adhering to the necessary data contracts. We turned our abstract patterns into concrete ones. These implementations were achieved using ActiveVOS and BPEL.

All this is still seen on a conceptual level. A next phase in the project will be the actual implementation of the language. If this is finished, MDO applications can finally be implemented using a real workflow language. They will gain all benefits such as evolution and reuse, customization, collaboration and distributive computation. They can then focus on what really matters: implementing new design iterations.

Appendix A

Optimus FEM Simulation

The input file has the following structure:

Nodes:
node Node_number pos_x pos_y
Materials:
material Material_number Density Cross-Section-Area Young modulus
Elements:
beam Beam_number Node_1 Node_2 Material
Single Point Constraint in X-Direction:
fixx Node_number
Single Point Constraint in Y-Direction:
fixy Node_number
Force in Y-Direction:
forcey Node_number Force_intensity
Number of eigenmodes:
nummodes Number
Viewing Parameter:
zoomfactor Number
END:
end

The structure of the output file is the following:

Mass = Mass
Node node_number: dx = dx dy = dy
Beam Beam_number: Sigma = Stress
Node Node_number: FX = Reaction_force_x-direction
Node Node_number: FY = Reaction_force_y-direction

Figure A.1 shows two possible instances of these files.

Input file

```
node 1   -9.6 0
node 2   -8.4 0
node 3   -7.2 0
node 4   -6 0
node 5   -4.8 0
node 6   -3.6 0
node 7   -2.4 0
node 8   -1.2 0
node 9     0 0
node 10   1.2 0
node 11   2.4 0
node 12  3.6 0
node 13  4.8 0
node 14  6 0
node 15  7.2 0
node 16  8.4 0
node 17  9.6 0

...

material 1 7800  0.004 2.1e11
material 2 7800  0.004 2.1e11
material 3 7800  0.004 2.1e11

beam 1 1 2 1
beam 2 2 3 1
beam 3 3 4 1
beam 4 4 5 1
beam 5 5 6 1
beam 6 6 7 1
beam 7 7 8 1
beam 8 8 9 1
beam 9 9 10 1
beam 10 10 11 1
beam 11 11 12 1
beam 12 12 13 1
beam 13 13 14 1
beam 14 14 15 1
beam 15 15 16 1
beam 16 16 17 1

...

fixx 1
fixy 1
fixy 17


forcey 9 -500000

nummodes 10

zoomfactor 15

end
```

Static Output file

```
+---------------+
| StatTruss 2.0 |
+---------------+

Results of this program can be used for demo purposes only
There is no guarantee of correctness of results

Results for problem in file pennsylvania.inp

Mass = 4535.12

Nodal displacements
   Node 1: DX = 0 DY = 0
   Node 2: DX = 0.000211459 DY = -0.00285464
   Node 3: DX = 0.000425175 DY = -0.00450769
   Node 4: DX = 0.000657583 DY = -0.00590886
   Node 5: DX = 0.000889435 DY = -0.00772455
   Node 6: DX = 0.0012832 DY = -0.00933992
   Node 7: DX = 0.0016766 DY = -0.0112068
   Node 8: DX = 0.00219061 DY = -0.0126869
   Node 9: DX = 0.00270412 DY = -0.0145284
   Node 10: DX = 0.00321764 DY = -0.0126869
   Node 11: DX = 0.00373165 DY = -0.0112068
   Node 12: DX = 0.00412505 DY = -0.00933992
   Node 13: DX = 0.00451881 DY = -0.00772455
   Node 14: DX = 0.00475066 DY = -0.00590886
   Node 15: DX = 0.00498307 DY = -0.00450769
   Node 16: DX = 0.00519679 DY = -0.00285464
   Node 17: DX = 0.00540825 DY = 0
...

Stresses
   Beam 1: Sigma = 3.75995e+007
   Beam 2: Sigma = 3.75994e+007
   Beam 3: Sigma = 4.08145e+007
   Beam 4: Sigma = 4.08145e+007
   Beam 5: Sigma = 6.9099e+007
   Beam 6: Sigma = 6.9099e+007
   Beam 7: Sigma = 9.01118e+007
   Beam 8: Sigma = 9.01118e+007
   Beam 9: Sigma = 9.01118e+007
   Beam 10: Sigma = 9.01118e+007
   Beam 11: Sigma = 6.9099e+007
   Beam 12: Sigma = 6.9099e+007
   Beam 13: Sigma = 4.08145e+007
   Beam 14: Sigma = 4.08145e+007
   Beam 15: Sigma = 3.75994e+007
   Beam 16: Sigma = 3.75995e+007
   Beam 17: Sigma = -7.28614e+007
```

Figure A.1: Example input and output files.

Appendix B

Simulation examples

B.1 Design of experiments

	Section1	Section2	Section3	Mass	Max_Stress	Max_Displacement
1	0.0047218907066115	0.0056119106196532	0.0034174799136123	4320.3119155238	8.5359657517615E7	0.014001426757173
2	0.0059849333387222	0.0044969845146242	0.0040115537122023	4899.121686197	1.0299282409241E8	0.013121691371846
3	0.0064794552307534	0.006036606510732	0.0063163623979763	7153.8849580283	7.6727651484336E7	0.0094466142480439
4	0.0022493714620442	0.0055930840585638	0.0045732976225632	4953.3987317306	1.6023291639147E8	0.015343259719144
5	0.0038852465522437	0.0064902915628671	0.0031260068118855	4041.6312290111	9.3325446219086E7	0.014866596139737
6	0.0030801361045184	0.0061086295904007	0.0059807815367093	6361.074775813	1.1698440807267E8	0.012014887984129
7	0.0048145792834586	0.0049456649646725	0.0049422835554309	5584.7253406181	9.3655870376752E7	0.012030217072772
8	0.0061380824672177	0.0060906705943375	0.0037090935795524	4840.7000581878	7.8643212242575E7	0.01247546702815
9	0.0021163276025156	0.0057180589156585	0.0032263125385546	3775.9012582127	1.7036588278555E8	0.018007422827372
10	0.0025746114003749	0.0046746839548519	0.0039154483489795	4325.0517219244	1.400099107393E8	0.01614231726009
11	0.0017669776352603	0.0036832424068277	0.0046949285278451	4769.1380226901	2.0399925551146E8	0.018692351547884
12	0.0050258157993193	0.0042795224606491	0.0022457002003207	3194.4538928173	1.2992694655104E8	0.018957114672166
13	0.0055779006222561	0.0050592712900769	0.0045250484951255	5349.0189464685	9.1549006589743E7	0.012053772986716
14	0.0036132819017608	0.0043746913885836	0.0028060860762702	3481.2825098222	1.0587903482523E8	0.01752617274649
15	0.0029826733888969	0.0052117126628223	0.0018339977679059	2636.630312019	1.591321158347E8	0.022575537409901
16	0.0060781274835677	0.0055260548850713	0.0042907255215172	5273.3207877324	8.3814066749565E7	0.011854148442909
17	0.0035367653515581	0.0054904574083658	0.0042255656088145	4831.9795237959	1.0190613287261E8	0.013571533957893
18	0.0053109790318767	0.0063246040246367	0.0051364661938986	5985.2762350957	7.3235213656466E7	0.010716792234872
19	0.0020234337146872	0.0058354668103638	0.0038723474227748	4337.4071501424	1.7815452933688E8	0.016933004765492
20	0.0037052951860403	0.0056654879905689	0.0039580430333457	4644.4473703072	9.7275837436488E7	0.013704412001471
21	0.0041778606060628	0.0016617241112234	0.0061620817424682	6175.9914733325	2.7874881962981E8	0.019133118742398
22	0.0059430203353557	0.0019595889163825	0.0031860055706629	3885.2394429359	2.3634159276745E8	0.020469959223428
23	0.0030219294549395	0.0041020950356454	0.001591414117475	2304.91027754	1.8341075001254E8	0.025718357813984
24	0.0027028745935002	0.0015503932025442	0.0048406001937743	4792.759425862	2.988013789359E8	0.022876418171785
25	0.0040068561225869	0.0024720872658862	0.0059308505177416	6041.6472463234	1.873777037061E8	0.015610287588243
26	0.0034227859352733	0.0024097145333198	0.0062436560558137	6219.1920517013	1.9223772590743E8	0.016164695029371
27	0.0020748435610229	0.001806034021724	0.0016888435653341	1985.951691898	2.5648860267077E8	0.032776320677081
28	0.0062995003153475	0.0021039350184807	0.0025420856778594	3394.903178877	2.2011570141025E8	0.021784289742673
29	0.0057473209460824	0.0021513828876711	0.0053054667353414	5721.6597798572	2.1528831963037E8	0.016315244852499
30	0.0032781339520276	0.0032059660797537	0.0053725106457323	5530.4831937039	1.4449368071982E8	0.015043387959539
31	0.0016308996865883	0.0036019207007258	0.0021485878622929	2524.2615444898	2.2123390055637E8	0.025957488985896
32	0.0042402076677299	0.0035541132554069	0.0060479690421645	6301.8959491283	1.3033117082057E8	0.012963486215653
33	0.0051860454591618	0.0044205753604457	0.0051671518052138	5776.0927504008	1.0477915294715E8	0.01206162011664
34	0.0043081543772111	0.0028447217054666	0.0046190833802952	4988.08259389	1.6282609129288E8	0.015617159389297
35	0.0017317857165474	0.0059446521510097	0.0044061667799694	4770.5864290422	2.0814161273372E8	0.017309055635101
36	0.0026764245102118	0.005145969955651	0.0032567175075113	3820.9815229658	1.3470679621738E8	0.016902033218694
37	0.0053943047038206	0.0023098794721255	0.0047080333530302	5167.1248732323	2.005164188684E8	0.016477003834761

38	0.0044078005150419	0.0025952305596317	0.0024367970918726	3076.0409943029	1.7845952738507E8	0.02134544767184
39	0.0016850584997563	0.0026767000425532	0.0054918463939037	5335.3551269796	2.1391052336487E8	0.020206813815814
40	0.002429368128459	0.0058583960374217	0.0037546275054965	4298.4031695272	1.483838919164E8	0.015937319900304
41	0.0040649670771769	0.0031711909613404	0.0026200605878442	3249.8233265628	1.4605331777108E8	0.019470157051435
42	0.0060341555563754	0.0042075983074981	0.0062600040921194	6829.5551084516	1.1008129687127E8	0.011028906952632
43	0.0029384078395462	0.0043460448444311	0.0021665587340622	2820.5809093797	1.3468975477874E8	0.02117836767283
44	0.0039657579452768	0.0041530137267637	0.001715622948481	2560.1227754013	1.7011118785102E8	0.023430681713114
45	0.0018829626896546	0.0034781205414882	0.0043496814899763	4462.7601150423	1.9144425628301E8	0.018913702916654
46	0.0056365692139058	0.0027699175119015	0.0055065124113283	5950.5244165698	1.67216005877714E8	0.014155958435607
47	0.0055058583959135	0.0015108147729671	0.0033615098400515	3921.2662229435	3.0654867015369E8	0.023164514859424
48	0.0045669288001632	0.0047488112545206	0.0041480385722212	4834.2208331773	9.7537393973295E7	0.0133641645176
49	0.0032462237956787	0.0027050993191432	0.0052180560486513	5334.2069301875	1.7124699738506E8	0.016336769690382
50	0.002616496950008	0.0032873717565758	0.0064125181302171	6345.4493675089	1.4093017418216E8	0.015192806288878
51	0.0042754823319544	0.0061676281935755	0.0056447979092158	6254.5257458333	8.4276789741245E7	0.010960035674486
52	0.0033600411174342	0.0033364869492123	0.0060583271180995	6154.2714333356	1.388417020476E8	0.014156562911435
53	0.0024736961342579	0.0045121257958174	0.0052841045363009	5482.0778761927	1.4568774758576E8	0.014810396643308
54	0.0037677347489002	0.0019040673836583	0.0020117577427686	2531.5838953405	2.4323611977097E8	0.026870961820497
55	0.0046095243106249	0.004038193159097	0.002741841657496	3536.2093555376	1.1469340964492E8	0.017324731545312
56	0.0052831041880677	0.0025165726573348	0.0034908367711409	4115.1304393578	1.8404151307538E8	0.017803662798895
57	0.0028282013904271	0.003095719126023	0.0057977102913685	5820.4347470369	1.4964865893688E8	0.015574403927151
58	0.0036957894505407	0.0029389603118389	0.0057090796892423	5855.3798720486	1.5761545106717E8	0.014827189419392
59	0.0063462010817624	0.003444017488196	0.0043910113362301	5163.2404229975	1.3448108024783E8	0.013748580103849
60	0.0031009589738253	0.0054306345796333	0.0061493646038044	6433.5238084819	1.1619842444791E8	0.01225004400748
61	0.0064130201990306	0.0029903991410673	0.0029289380845385	3849.5590812652	1.5487064660026E8	0.017610995324151
62	0.0031865634530415	0.0048902839224156	0.0029792587788578	3626.8381972427	1.1315080190578E8	0.017017611293206
63	0.0019651839091251	0.0016083870247552	0.0044792270786039	4374.5162709948	2.8807058190988E8	0.024762966537175
64	0.0048737405070644	0.0046025075785123	0.0023706984970343	3317.2618667563	1.2307271509919E8	0.01809471282868
65	0.0039191813003786	0.0053508062276675	0.004196123505194	4847.7083296503	9.1962155558476E7	0.01332618495266
66	0.0043503494315044	0.0062213650262157	0.0027822900100442	3781.5910532244	1.0485859000612E8	0.015667922491977
67	0.0051084132386397	0.0030119944887914	0.0064922636528451	6756.6055558274	1.5378244362215E8	0.013103736584328
68	0.0019312215590821	0.0062584524910565	0.0049704410723731	5327.1429682856	1.8662215323643E8	0.015625520817147
69	0.0038441110135217	0.0047985851139083	0.0019474738161959	2817.2388691697	1.498447648642E8	0.021065720086236
70	0.0058815931921968	0.0053434372830439	0.00380242937067	4798.2593317653	8.6677628813593E7	0.012839234205753
71	0.0034745973176991	0.0026178188968647	0.0017715557911084	2360.1255540452	1.7691753765624E8	0.026155430972669
72	0.0044989244869512	0.0049502119272653	0.0024542227752009	3373.4507429144	1.1888316620067E8	0.017639762623187
73	0.0056639497578895	0.0035329738529206	0.003002640526406	3863.3814108509	1.3109057841928E8	0.016649576325539
74	0.0052480397099984	0.0038151052151566	0.0054165098894542	5933.250889091	1.21407910125 91E8	0.012490254086551
75	0.0057598322068587	0.0020187944108707	0.0016190688089606	2501.3810584163	2.2937534414133E8	0.028333624396987
76	0.001585103996753	0.0039315866471783	0.0036021468524975	3819.5465596537	2.2746561989347E8	0.020960857343266
77	0.0063803168184582	0.0052611569475544	0.0050492289123747	5948.2329260755	8.8034883890487E7	0.011001105870041
78	0.0049430588621981	0.0039913384653541	0.0058674440909196	6299.9737419894	1.1604978409271E8	0.012049569152335
79	0.0046512282952643	0.0017303524341281	0.00555757624606	5728.8131962711	2.676838265495E8	0.018820773934202
80	0.005071308182997	0.004587801346714	0.0048546872720524	5506.1505025243	1.0095982920172E8	0.012289383190757
81	0.0047653450540666	0.0031267600178735	0.0025849565238947	3319.1050135692	1.4812221995161E8	0.019224797699585
82	0.0035912647011311	0.0020984277767624	0.0047770029079234	4932.9849064001	2.2074466723608E8	0.018465662194744
83	0.0049833728052268	0.0017786128284231	0.003518917009664	4010.5015707586	2.6040227990655E8	0.021104647451107
84	0.0027849080729763	0.0022825462433857	0.0028750335868942	3178.5821496047	2.0294278396509E8	0.022455653204152
85	0.0028755641836179	0.0022291893181471	0.0020861242233197	2499.7511159245	2.0778312166602E8	0.025981021769806
86	0.0041423878799298	0.0059821351433573	0.0015318911061073	2635.3600187131	1.9053212487467E8	0.02370947452245
87	0.0023246898651724	0.0040635365407715	0.0023035925046086	2815.6626166512	1.5517171085866E8	0.021988042975573
88	0.0021959677239598	0.0038721711927167	0.0030763275620073	3446.8094196984	1.6420473773161E8	0.01973185716448
89	0.0062287412913167	0.0037760656517738	0.0033330886762872	4263.1686919882	1.226514399433E8	0.015210812316094
90	0.004549908887966	0.0051595257759045	0.0018735567016685	2899.8020012889	1.5575671462103E8	0.020795008997713
91	0.0022682268093572	0.0050170862197365	0.003572669252024	4020.0159952403	1.5893916146998E8	0.017243290250209
92	0.0015063596435947	0.0033616053872853	0.0056546795032644	5528.3705638459	2.3927781275419E8	0.019705803358953
93	0.0025375531604103	0.0064178879518725	0.0058407998223816	6193.3118861843	1.4200461114308E8	0.012975797947281
94	0.0061756651434865	0.0057866652753633	0.0022830942858015	3571.0877649544	1.2779422120359E8	0.017101551118431
95	0.0054777009428764	0.0018768833338182	0.0019862905498902	2762.412372568	2.4673501085495E8	0.026139007793168
96	0.0054388971328139	0.0037457424460049	0.0026771333640648	3570.7662084516	1.2364278980356E8	0.017506096839084
97	0.0058483361157678	0.0063918123827382	0.004088372348965	5161.6136141429	7.2461841238834E7	0.011769595325468
98	0.0018067440586349	0.0048337692830599	0.0050718284341139	5234.2086415865	1.9948703715752E8	0.01691701073534
99	0.0023885448572369	0.0023511658943353	0.0036912440772305	3837.1222851703	1.9704127059776E8	0.021000169689927
100	0.003323270539027	0.0028913635946165	0.0063617458065463	6361.9580316259	1.6021701763102E8	0.01487911592233

B.2 Optimization simulation

B.2.1 Sequential quadratic programming (SQP)

	Label	Iter	Annot	Section1	Section2	Section3	Cost
1	1	1	START	0.0040	0.0040	0.0040	13666.574134256
5	2	1	LINE	0.00387072146742	0.0044832121285052	0.0032479835625211	11948.399173603
9	3	2	LINE	0.0034828290809253	0.0055789402658978	0.0015	13216.895507244
10	4	2	LINE	0.003724768718338	0.004895503112982	0.0025902676207219	11324.730590714
14	5	3	LINE	0.0036107111385933	0.0048179474514431	0.0026661189047619	11259.3950441
18	6	4	LINE	0.002895430736823	0.0043357787747296	0.0029318779152402	12054.414833699
19	7	4	LINE	0.0035016486491951	0.0047444287082172	0.0027066405508059	11207.206626214
23	8	5	LINE	0.0024736878901246	0.0039527568635369	0.0029324563302138	12787.935907847
24	9	5	LINE	0.0033869027918103	0.0046560585405978	0.0027318471795518	11153.369554532
28	10	6	LINE	0.0024235937612601	0.0037446325609636	0.0028335494446179	12699.222831566
29	11	6	LINE	0.0032779149603133	0.0045529407061957	0.0027433536732899	11260.412415529
30	12	6	LINE	0.0033686519779217	0.0046387907008911	0.0027337740265478	11158.458713512
31	13	6	LINE	0.0033810673967736	0.0046505374352524	0.0027324632569205	11150.482281386
35	14	7	LINE	0.0022873705629476	0.0033724712519303	0.0027288976662012	12798.851862866
36	15	7	LINE	0.0032487966002003	0.0044959691610168	0.0027320320373081	11264.701716375
37	16	7	LINE	0.0033557596163418	0.0046209634094854	0.0027323807503389	11166.779886623
38	17	7	LINE	0.0033753236637441	0.0046438254555482	0.0027324445316206	11149.937436912
42	18	8	LINE	0.003362602158797	0.004149122215161	0.0027271607928734	11242.305298573
43	19	8	LINE	0.0033718448495253	0.0045085442389079	0.0027309996439631	11122.290136437
47	20	9	LINE	0.0033582908996306	0.0015	0.0027127726465834	19320.983052436
48	21	9	LINE	0.0033704894545358	0.0042076898150171	0.0027291769442252	11192.104701143
49	22	9	LINE	0.0033715138367844	0.0044350699762428	0.0027305545066852	11106.101120354
53	23	10	LINE	0.0033529383351427	0.0015	0.0027102591286759	19315.340068961
54	24	10	LINE	0.0033696562866202	0.0041415629786185	0.0027285249688843	11253.877796993
55	25	10	LINE	0.0033712297609061	0.0043901838354915	0.0027302441286699	11096.239351341

B.2.2 Self adaptive evolution (SAE)

	Iter	Section1	Section2	Section3	Cost
1	0	0.0040	0.0040	0.0040	13666.574134256
2	0	0.0039664924353431	0.0063140169564975	0.0026851059253275	11679.671853676
3	0	0.0056843640632616	0.0022500448610927	0.0015280680481352	13661.603393911
4	0	0.0050310635903649	0.004892878697433	0.0058658015886722	16288.846317556
5	0	0.0016537582746979	0.0018091547535771	0.0030542663858666	17149.1676313
6	0	0.0033801403626681	0.0027155305346561	0.0021924458945203	12651.778552606
7	0	0.0041485630285097	0.0035337583098793	0.0063362195815706	18121.865689216
8	0	0.0020633086660976	0.0051508250822307	0.0055315871570949	18474.746044557
9	0	0.0063353859383511	0.0037917959228606	0.0038740757486391	14354.850664281
10	0	0.0029216416049595	0.0059444452220342	0.0040427178810609	14183.790936008
11	0	0.0053218005468108	0.0055347035332242	0.0044178908815303	13645.620814869
12	0	0.0032006845468719	0.0031863747622126	0.0049341911119402	16102.578657267
13	0	0.0043587620685088	0.0041022147622555	0.0024029093326177	11411.633289197
14	0	0.0058718738725421	0.0020988539927728	0.0052880815064297	20518.993460141
15	0	0.0023911944035955	0.0045791397720983	0.0033891445117202	13870.546632921
16	1	0.003771717888378	0.0052824138514702	0.0049201902566185	14500.23353387
17	1	0.003672090355315	0.0041228883075974	0.0019251891128596	11763.19360205
18	1	0.0042978222764961	0.0032961909804075	0.0018377343653962	11935.746954485
19	1	0.0046586588303646	0.0052011675817011	0.0028956270687829	11633.95435035
20	1	0.0046678412326043	0.0059107551538981	0.0031986083798149	11867.586371176
21	1	0.0054012317603343	0.0056544581650032	0.0050895344380193	14720.318245008
22	1	0.0048006787394163	0.0026108035816132	0.0049192273126007	17816.037078731
23	1	0.0037211856687802	0.0045458720844512	0.0016525921627509	12521.173480905
24	1	0.003374401929728	0.0051943974212098	0.0055336057705361	15872.087218181
25	1	0.0053375663841068	0.0060528213095073	0.0034845008058722	12228.389957754
26	1	0.0045203127072901	0.005213515107142	0.0018454952469079	12395.339786614
27	1	0.0042579971398675	0.0043607603662939	0.0016126493213172	12759.987647563
28	1	0.0041016568987404	0.00542647665325	0.0050028909920106	14422.378383505
29	1	0.0055374498221968	0.0055452824462244	0.0040632916899077	13118.80412783

30	1	0.004999752329906	0.0052636357078763	0.0057868634481827	15936.723451741
31	2	0.004907740295055	0.0059251612481802	0.0046251106780183	13710.744675728
32	2	0.0043262394851845	0.0055683619064738	0.0055778566674418	15260.912560744
33	2	0.0056068935490152	0.003821966982442	0.0030769191315083	12801.958529058
34	2	0.0037387341343106	0.0053554649251665	0.0028385168924143	11402.340149015
35	2	0.0029915777998103	0.0064884778345294	0.0038714343243077	13912.474199447
36	2	0.005563931864618	0.005257537508099	0.0035071549157101	12350.904437895
37	2	0.005605865947078	0.0024923505722276	0.0048593470659194	18290.649228642
38	2	0.0052122726837231	0.0056932577097439	0.0036029156409083	12206.165853492
39	2	0.0052301900208149	0.0059720918739961	0.0028438363171466	11957.454608308
40	2	0.0029424948478647	0.0057367577480223	0.0043993283077777	14692.416604735
41	2	0.0030645935815782	0.0050513709102402	0.0057822213801875	16637.301874083
42	2	0.0047126133865032	0.0059573864616655	0.0038947813043803	12442.985620785
43	2	0.00495629319591	0.005686243569199	0.0021252451103359	12164.067897475
44	2	0.004115778734538	0.0041951569678438	0.0028604005089024	11630.334450161
45	2	0.0056917814851245	0.0055446985498355	0.0044683285319546	13828.745076802
46	3	0.0027662037537441	0.00469126510229	0.0018117258699759	11861.236681537
47	3	0.0047358306217777	0.0057666459749183	0.0035534045623424	12038.279757016
48	3	0.0046189083387814	0.0027867424432718	0.005284894674557	17911.060184022
49	3	0.0033686268518189	0.0051913933294944	0.0020863949120178	11657.115210922
50	3	0.0052180660897842	0.0064799273466893	0.0025151272297924	12113.28113566
51	3	0.0055640237148969	0.0064369715252051	0.0029675866482428	12167.278777322
52	3	0.0039709355282057	0.0049724820957044	0.0040068658672039	12882.815858851
53	3	0.0056994436779791	0.0053165707936	0.0029581296453037	11962.357206687
54	3	0.0032158355576543	0.0055037117022635	0.0020478418375369	11733.806894373
55	3	0.0054472963043862	0.0058645645081317	0.0029345838456525	12005.26221805
56	3	0.0045894541790451	0.0051070177289532	0.0020472270296522	12037.547190526
57	3	0.0049539305343586	0.0047907702906488	0.0024194307792286	11719.081332179
58	3	0.0062695159867837	0.0036438220350247	0.0042505390110251	15141.624862833
59	3	0.0057449610524508	0.0049715181421046	0.0027373976622831	11888.000888081
60	3	0.0063966353366343	0.0049323508890063	0.0031333167068529	12157.185246849
61	4	0.0041071602135452	0.0049475937320875	0.0038228555263284	12634.006228015
62	4	0.005237597301699	0.0042707894152202	0.0017899836916143	12519.659681518
63	4	0.0050685011610688	0.0058635961588909	0.0029788387256314	11905.629368681
64	4	0.003678516265535	0.0054829087344252	0.0020925709307152	11799.540088658
65	4	0.003866199790326	0.0062932370779717	0.0039126526439551	12986.214341116
66	4	0.0057874413099549	0.0046898292199559	0.0021701303564354	12133.172076694
67	4	0.0058160405110631	0.003990236048343	0.002857772241462	12312.155521349
68	4	0.0029266393554126	0.0061254420006783	0.0024779935734565	11645.307294213
69	4	0.0034387736772954	0.0056319155531277	0.0036188190428463	12749.89592526
70	4	0.0054976880625701	0.0050634171483839	0.0022973297626808	12013.11333107
71	4	0.0061358657733624	0.0047448389874579	0.0027822859660462	11947.85540075
72	4	0.0061174223439562	0.0049346362718215	0.0023366885471623	12131.223695242
73	4	0.0050427714252528	0.0052489008645432	0.0032783415413939	11864.445910709
74	4	0.0033754054530013	0.0041195597241087	0.0024524291169519	11110.764190508
75	4	0.0055720395273224	0.0044337968110192	0.0015232527180454	13468.138499852
76	5	0.0031187134054929	0.0057180018641646	0.0022557382074844	11516.139831549
77	5	0.002207362169018	0.0035903066241249	0.0026224510153251	12899.96262131
78	5	0.001922719470275	0.0063702827720188	0.0021655804894091	13733.823527876
79	5	0.0037662924081209	0.0052561948183118	0.0018137721274627	12260.967987289
80	5	0.004100700983578	0.0059023937394352	0.0018724922506293	12369.180808712
81	5	0.0025662611815585	0.0058194866479023	0.0022952375497835	11939.202808824
82	5	0.0028649051881396	0.0032866351259116	0.0031064890380994	12828.719871336
83	5	0.0031857564353685	0.0044357938383955	0.002781179171970 3	11411.245288377
84	5	0.0025726162323399	0.005915236391061	0.0021177206274412	11655.303116873
85	5	0.0041756848896289	0.0057117472334123	0.0024491481884563	11681.918401293
86	5	0.0034000944047412	0.0062949076595021	0.0029413056278705	11821.398625209
87	5	0.003017766423384	0.005216667267509	0.0017424359030225	12213.717754534
88	5	0.0019150925601181	0.0045432976610881	0.0036772365009499	15850.761946287
89	5	0.0034211390075545	0.006485053027761	0.0021072870649807	11924.129385373
90	5	0.0032719342293752	0.0064296848485747	0.0023554393769527	11635.725963806
91	6	0.0026940727017477	0.0041893315654535	0.0035851133579427	13448.221602624
92	6	0.0034968331648707	0.0054673841887322	0.003451193705641	12378.89849875
93	6	0.0035788764616282	0.0059975838027854	0.0017632277696839	12487.818350647
94	6	0.0035659537190699	0.0060275969069643	0.002243752834606	11720.021797808
95	6	0.0037080423878972	0.0039915913447942	0.0028960872340782	11777.627188867
96	6	0.0034150434224422	0.0044911044669656	0.0023361389515541	11271.608852197
97	6	0.0035215606416343	0.0056209454815869	0.0027995467131955	11397.063530347
98	6	0.0033344110804942	0.0061426864007195	0.002492278312995	11517.069319775
99	6	0.0039261714146258	0.0061540563941521	0.0030459239712641	11660.435743791
100	6	0.0032252175436459	0.0060364122810158	0.0026411274856413	11476.373974891
101	6	0.0027843686262168	0.0060817949132744	0.0023023290075898	11584.549775803
102	6	0.0029039366668462	0.0062343671490109	0.0023624401932802	11515.150692806
103	6	0.002817876472683	0.0032226690959678	0.0028244353944679	12460.882951437
104	6	0.0030602093777925	0.0040685167724054	0.002539296859634	11103.917523071
105	6	0.0036362779657707	0.0059715828634319	0.0031388910736972	11840.007559605
106	7	0.0033976814665441	0.0048473246489089	0.0025589710531116	11231.04619706
107	7	0.0031918133981101	0.0046885246958597	0.0021347737475642	11438.722260779
108	7	0.0029860089189314	0.0054733412975453	0.0019324330204434	11847.304559481

109	7	0.004229424828774	0.0057302322591659	0.0022583372893364	11830.536849812
110	7	0.0033988098796171	0.0042632869880174	0.0025664016237476	11103.148125666
111	7	0.0026767540966112	0.0043379632102372	0.0023525064281484	11489.949030821
112	7	0.0030471904519149	0.0057674901365745	0.0026528287199701	11674.717578264
113	7	0.0038871150581652	0.0045934703162787	0.0017963205690496	12192.40176285
114	7	0.0034762620527495	0.0054637958287157	0.0028885246544576	11474.947333914
115	7	0.0032850463701307	0.0045184942722441	0.0027539517479913	11261.902037731
116	7	0.0041352941712436	0.0033189474199511	0.0018836772589839	11799.688274572
117	7	0.0033265275438609	0.0034435418955326	0.002424809457683	11589.189758126
118	7	0.0039555976714654	0.0038213366029631	0.0019196090804632	11788.568010251
119	7	0.0038390219794699	0.0039845179013221	0.0026913682636469	11485.5103725
120	7	0.0030618320608902	0.0054849196355992	0.0022700057554214	11437.479951794
121	8	0.0036877875189627	0.0046344813496373	0.0028753364832499	11235.108642187
122	8	0.0038485606121402	0.0049563338881991	0.0031959794388436	11534.457454355
123	8	0.0035723465126185	0.0042228032809254	0.0024126285992472	11209.672739057
124	8	0.0033896745366401	0.0046534991118694	0.0027050769211212	11156.115103174
125	8	0.0032931473086477	0.0052036708112438	0.0016856146524486	12443.940262883
126	8	0.0031232824481033	0.0047626852042736	0.00302876388687	11970.145057613
127	8	0.0043365453504498	0.0046838401115232	0.0027376649744925	11427.509509586
128	8	0.0035029203073475	0.0039288259377883	0.0022444711515689	11249.117553881
129	8	0.0034541801344036	0.0044789509398806	0.0028234101503441	11178.778644469
130	8	0.0033666488960487	0.0044606169090667	0.0025532346024787	11140.755708494
131	8	0.0037930306348444	0.0045001746346911	0.0031405393389093	11736.269793846
132	8	0.0036521330615963	0.0040672873712361	0.0026055500482747	11205.30985737
133	8	0.0030971183353978	0.0047434348643011	0.0040907451135545	13746.02078994
134	8	0.0035786698960604	0.0051472989619044	0.0027601707892269	11311.816373499
135	8	0.0034327780398854	0.0052735758011392	0.0028324995819748	11387.474166124
136	9	0.0031831497890396	0.0046589779780852	0.0027508081473666	11412.796353688
137	9	0.003356917975043	0.0045572822172959	0.0026064539333174	11143.623575926
138	9	0.003651889197825	0.0042178431462797	0.0027744645974245	11336.636802594
139	9	0.0031871340015508	0.004608663619602	0.0023710555877159	11208.953365791
140	9	0.0035427906249567	0.0045134730574054	0.0023155601785112	11327.583264925
141	9	0.0036002625085745	0.0043758941469728	0.0026824635719787	11158.666417252
142	9	0.0036354909438213	0.0047500505819247	0.0026699965766212	11251.14486667
143	9	0.0033977493718084	0.004504933029382	0.0026583212471354	11133.010930039
144	9	0.0032315668718309	0.0043163778560981	0.0023449165657106	11175.923288452
145	9	0.0034722236250439	0.0044231433107128	0.0025218300775834	11173.543260807
146	9	0.0033206361054144	0.0044990613497667	0.0028273919775404	11336.601538357
147	9	0.0034076251300051	0.0045425680653106	0.0028800783754874	11335.122636762
148	9	0.0031468001709636	0.0044351413677155	0.0026074970124432	11176.618024182
149	9	0.0034819223309327	0.0045229326984321	0.0025365780584127	11192.430795035
150	9	0.0032325706269579	0.0046408305270407	0.0023427907149703	11247.625226323
151	10	0.0033428831746567	0.0045432506621269	0.0023705233249849	11239.221534051
152	10	0.0033402558277267	0.004571109767995	0.0025824421268853	11148.235612782
153	10	0.0035261586377626	0.0044232824203384	0.0024765263701075	11207.515058348
154	10	0.0034478337129223	0.0044613318048978	0.0028294011865494	11191.480925509
155	10	0.0033821780505777	0.0043883284994198	0.0026529653671286	11104.420287439
156	10	0.0033824141270673	0.0046199192896401	0.0026551151679711	11154.025575159
157	10	0.0033731653669067	0.0044840220559851	0.0028830579961491	11365.22164441
158	10	0.0034520104249276	0.0044934979953601	0.0027182057328354	11137.902422767
159	10	0.0038218794575507	0.0043777175048011	0.0026142055924002	11234.526151545
160	10	0.0034976483226711	0.0044824644825695	0.0027867498405931	11144.906554729
161	10	0.0033438770673761	0.0044013004459719	0.0026279884077143	11101.336286032
162	10	0.0033534751300385	0.0046595972744943	0.0026650422947477	11152.698755376
163	10	0.0033636579105236	0.0045324498350398	0.0026253071147645	11135.784263259
164	10	0.0034700573531658	0.004496519122364	0.0025882020493479	11167.257485266
165	10	0.0032021364167727	0.0046912413498143	0.0025085678287873	11160.101766971
166	11	0.003323547035537	0.0044938431583922	0.0026441635796253	11112.23350495
167	11	0.0034357658558887	0.0046251510691761	0.0026963262849751	11164.058788138
168	11	0.0032151126375486	0.004519203656398	0.0025065639947672	11127.482844402
169	11	0.0034061769123547	0.0043700925340397	0.0026361377548075	11110.512968148
170	11	0.0035136720363075	0.0043639478972076	0.0025597680283084	11159.388261442
171	11	0.0033483803113432	0.0043912061782741	0.0027007398004034	11088.390897881
172	11	0.0032831249377763	0.0045423550609833	0.0025985478997384	11121.537060523
173	11	0.0033008345435486	0.0043909177236743	0.0026730128601517	11082.899266291
174	11	0.0033451748801482	0.0045388194589745	0.0025362519579188	11157.306892872
175	11	0.0034391359744986	0.0045203707705857	0.0026335281087001	11152.765190649
176	11	0.0032109454813769	0.0043249426615987	0.0025013444926999	11086.539861081
177	11	0.003367966366461	0.0045139583884338	0.0026546573448521	11127.180138086
178	11	0.0033498785218552	0.0042989086617158	0.0026990278329451	11069.119728365
179	11	0.0033753132511236	0.0043717883476572	0.0026299295355297	11103.446621033
180	11	0.0032522942760005	0.0045421535383491	0.0027781618833004	11346.12488289
181	12	0.0033079912346169	0.0044818156436503	0.002631145424891	11107.873071834
182	12	0.0031937997975509	0.0044841525400001	0.0023497998652845	11198.058737671
183	12	0.0031051531121043	0.004261155386412	0.0026461464707677	11258.442839766
184	12	0.003229878514958	0.0043051528956199	0.0025971100138712	11055.722460147
185	12	0.0030549099285457	0.0042092715537748	0.0028361089085061	11628.961266693
186	12	0.0033260460486894	0.004338398116297	0.0026565235123495	11077.151473105
187	12	0.0029244478239914	0.004212535999965	0.0027116011011017	11620.078999084

188	12	0.0033850399706137	0.0043274439694657	0.0026914668490167	11086.150019459
189	12	0.0031915874490111	0.0043879697447582	0.002850845319405	11507.879642704
190	12	0.0032977933921817	0.0042641517368357	0.0026128433214207	11062.171486892
191	12	0.0030249525502464	0.0044387866231581	0.0022352028529862	11232.442930881
192	12	0.0032950235860127	0.004393974432357	0.0026620895531273	11079.40312368
193	12	0.0031090696624498	0.0042147814898645	0.0023974753549691	11085.712079381
194	12	0.0031910929560029	0.0044238798550818	0.0026156716684758	11130.145957722
195	12	0.0034569610335514	0.0042346688960408	0.0023866266211616	11194.864091419
196	13	0.0033617679144725	0.0042213536593048	0.002692665898888	11116.862707004
197	13	0.0033348937567984	0.0042760945571387	0.0026011140437139	11078.118205897
198	13	0.0033209840509606	0.0043231503688361	0.0025489620542464	11099.619347111
199	13	0.0033323453008517	0.0043200393040242	0.0027382004226387	11138.115587586
200	13	0.003367776312584	0.0043607020372246	0.0025406243396917	11123.772971106
201	13	0.003220714558139	0.0042987582312878	0.0026903079751503	11188.366093784
202	13	0.0033455244747168	0.0043521513205567	0.0025290127075241	11119.760494029
203	13	0.003246023036273	0.0042789263230594	0.0026253987580674	11047.853500139
204	13	0.0032523746372578	0.0043783064221435	0.0025422642304124	11094.376779178
205	13	0.003313618660881	0.0042492879317372	0.0026028815676563	11065.877199463
206	13	0.003305520136363	0.0045075333235825	0.0026355085446899	11111.810294338
207	13	0.0032972579930754	0.0042908073834421	0.0026428398410205	11061.289023279
208	13	0.0034977378054774	0.004392394136189	0.002518809956131	11175.27046931
209	13	0.0033769541004564	0.004310501449591	0.002649202493125	11086.856081316
210	13	0.0032968822797035	0.0042919294017419	0.0026397929801665	11062.024698276
211	14	0.0032854185353884	0.0042462994338711	0.0026285896764061	11051.265335188
212	14	0.0032585586555333	0.0043172162091233	0.002629193802	11058.829485898
213	14	0.0033726929958249	0.0043224834133213	0.0027336059041189	11096.124484648
214	14	0.003339217085324	0.0042791694298679	0.0026571328702179	11068.000546855
215	14	0.0033870274641245	0.0042968519080081	0.0025653011935623	11107.390094401
216	14	0.00333504436055	0.00434776783161	0.0026246140986805	11088.034728689
217	14	0.0033140135487118	0.0043130503216649	0.0025575592028624	11092.667028823
218	14	0.0033001301331744	0.0042995538745102	0.0026201538331246	11068.767393812
219	14	0.0032464241580663	0.004256499860301	0.0025937709029211	11050.788930176
220	14	0.00314758239397	0.0042323723456528	0.0025447947965116	11032.429755643
221	14	0.0032421247101171	0.0042878175883792	0.0025868371367315	11058.198210142
222	14	0.0033127118099159	0.0043297479166941	0.002622371341923	11078.334067015
223	14	0.0033732621616224	0.0042910455431761	0.0026575593227603	11080.112086055
224	14	0.0033282980545905	0.0042752430827912	0.0026093319339647	11074.029095027
225	14	0.00330912274029	0.0042549245625502	0.0026979779593835	11084.899132119
226	15	0.0032121093180925	0.0042570926749249	0.002690141893182	11189.884393978
227	15	0.0032822006885734	0.0043159733859145	0.0024678970955024	11119.357546444
228	15	0.0032614330856469	0.0043695542462037	0.0026919591879831	11156.333020044
229	15	0.0030803657629662	0.0041941936673267	0.0025906410498952	11187.023676625
230	15	0.0031151217122381	0.0042751734228305	0.0026732354561113	11292.396412505
231	15	0.0031525789986776	0.0041476430061858	0.0026361448838877	11154.114996965
232	15	0.0031886278156707	0.0042165678698651	0.0024641673749563	11073.222426438
233	15	0.0030956014864987	0.0042252525939329	0.0025693539147405	11137.726763514
234	15	0.00334872824747	0.0042779350017653	0.0026186852119295	11078.181816779
235	15	0.0033086374844268	0.0042285180525287	0.0026582836853857	11048.235145817
236	15	0.0032765168551364	0.0042920059774818	0.0026184586486443	11060.850805876
237	15	0.0032931299060896	0.0042693819663383	0.0025151105631945	11092.322297124
238	15	0.0033238288776489	0.0041383985747733	0.0025049132522529	11076.790341419
239	15	0.0031844046579223	0.0042253300082633	0.0024662048426055	11072.960670605
240	15	0.0031901964839422	0.004312940258233	0.0027025287463014	11249.981849437
241	16	0.0033688188128587	0.0042645100960289	0.002722344761928	11128.109531276
242	16	0.0032683076070618	0.0042008286154012	0.0024867236724926	11082.155586502
243	16	0.0031952296405828	0.0041922900990546	0.0025525927305859	11034.659669054
244	16	0.0033110385891441	0.0042586763268353	0.0026210284683178	11062.841704606
245	16	0.0031331041496339	0.0041564383608824	0.0025402385482717	11024.364346702
246	16	0.0031581790026936	0.0042957319949565	0.0024448951580245	11091.086352336
247	16	0.0032473523017161	0.0042503884838317	0.0025681647686717	11057.009700989
248	16	0.0032701550139865	0.0043104309549035	0.0026661349874997	11090.693284498
249	16	0.0032465476167258	0.0042677924958469	0.0026292155824582	11049.507379312
250	16	0.0032788015869868	0.0043306622575668	0.0026746534727633	11098.685422372
251	16	0.003281573333614	0.0041455156551593	0.0023585752185554	11143.957607627
252	16	0.003348587539809	0.004267177250671	0.0026394788665501	11071.370233168
253	16	0.0033141380607356	0.0043029908889285	0.0026495491735568	11067.412158213
254	16	0.0033032602880339	0.0042346796901332	0.0026743945336528	11057.9620856
255	16	0.0033499978776611	0.0042149679482714	0.0024806624259011	11110.954238332
256	17	0.0033090540371024	0.0043164259870956	0.0026960799242555	11095.109509272
257	17	0.0031713465017704	0.0042734953124991	0.0027046080181607	11269.132141544
258	17	0.0032566523361113	0.0043709303999208	0.0027095226787732	11191.254247323
259	17	0.003152465726939	0.004316018830383	0.0027155372352943	11320.859489983
260	17	0.0030970345097321	0.0041853417652831	0.002519293285426	11045.000172112
261	17	0.003240080955341	0.0041736426161953	0.0027350782171025	11210.951423985
262	17	0.0030683018695546	0.0040318251272595	0.0026300163520084	11233.531575321
263	17	0.0031066651955323	0.0042310383175195	0.0026708715615395	11290.484530014
264	17	0.0029964338059734	0.0041897882606382	0.0027339182491343	11541.821035542
265	17	0.0031540371022963	0.0041967713631625	0.0025954443650369	11095.970719202
266	17	0.0031202289731786	0.0043214405935342	0.0028134801291393	11525.711205694

267	17	0.0031270279292685	0.0043275641599728	0.0025732365193304	11123.504425389
268	17	0.0031649853838495	0.004089779238955	0.0025153103885194	11017.301951701
269	17	0.0031506431193942	0.0041461468865515	0.0025861507322803	11074.272808735
270	17	0.0030604444705529	0.0042672749839423	0.0025824165760716	11217.162319108
271	18	0.0030259867151635	0.0039862387314248	0.0025894666103045	11217.511703686
272	18	0.0031649525432344	0.0041187179498702	0.0025718040861102	11026.091966347
273	18	0.0030277134975714	0.004072812221605	0.0026495350559722	11332.24841183
274	18	0.0031746874274002	0.004118798893856	0.0025163242808576	11025.909661381
275	18	0.0031944676164299	0.0042182208490375	0.0024062423048339	11105.48452178
276	18	0.0031136078531405	0.0041497125144999	0.0025591373881117	11080.136098387
277	18	0.0030442485627113	0.0041644689675366	0.0025258146735928	11125.127956589
278	18	0.0030788242326927	0.0042063158672836	0.002485579687229	11030.271732805
279	18	0.0031801572042371	0.0040465780930765	0.0026158617078097	11110.08728305
280	18	0.003223470885071	0.0040977202597125	0.0025382917993338	11027.09565951
281	18	0.003052832234196	0.0042505628628539	0.0025991209803031	11251.757327611
282	18	0.0031367131439627	0.0042198344230001	0.0025962865879828	11125.207546386
283	18	0.0030881920661952	0.0040975332377066	0.0025089267082916	11021.247005732
284	18	0.0031148571263681	0.004096095866444	0.0026220261149884	11170.125435477
285	18	0.0031587432576062	0.0041581939567614	0.0026054199430941	11097.872386223
286	19	0.0031459426752966	0.0040829487463077	0.0025571437975498	11019.246344192
287	19	0.0031339642986898	0.0041079554760312	0.0024785470879317	11027.755357624
288	19	0.0031500657585683	0.0040169489946355	0.0025915413285245	11092.351339968
289	19	0.0032212976302651	0.0041148881938702	0.0025652612739095	11021.321343817
290	19	0.0030615384122558	0.0040869473917924	0.002544211283557	11114.077115248
291	19	0.0030861752654709	0.004025301437321	0.0026130480354388	11179.37701017
292	19	0.0031003726304399	0.0041337561492127	0.0026240975354341	11201.33987433
293	19	0.0031004419304945	0.0041343506016232	0.0024372141565288	11043.90707527
294	19	0.003105749149809	0.0041085304191286	0.0025828417535256	11120.848659623
295	19	0.0031269178669442	0.0040856035195222	0.0025206978379285	11003.583890692
296	19	0.0029997695559766	0.0041559604499406	0.0024720968554298	11099.854716592
297	19	0.0031812798029547	0.0041111807715228	0.0025220757603146	11023.924548759
298	19	0.0031077535378065	0.0041026882685795	0.0026316265725189	11196.939803576
299	19	0.0031534609594818	0.0041526945073144	0.0025521606017722	11016.187597139
300	19	0.0031908790774428	0.0040855291379236	0.0026003273137409	11047.947365791
301	20	0.0031946468455085	0.0042505417680931	0.002441416089819	11093.420351439
302	20	0.0031470411090625	0.0040616913682858	0.0025889623735913	11065.44590201
303	20	0.0031372635888218	0.0040651369201561	0.0026340913497121	11153.216152507
304	20	0.0031644681825843	0.0041826600767115	0.0025786752395426	11051.771706805
305	20	0.0030851537349646	0.0041201572133928	0.0025043294793746	11022.775645587
306	20	0.0031451041557925	0.0040952809755662	0.0025647235310798	11035.451102822
307	20	0.0031865431031101	0.004093456934284	0.0025625798567924	11007.707154488
308	20	0.0031480808878488	0.0041029432707453	0.0024955422196352	11023.312822035
309	20	0.0031213382645944	0.0041243008054309	0.0025269067090477	11011.326407015
310	20	0.0031155760112455	0.0041762643738486	0.0024893292314582	11032.562397324
311	20	0.0032113779021823	0.004160903089894	0.0024921639430799	11055.123408448
312	20	0.0031266583578349	0.0040761370424311	0.0025425526412384	11019.487773787
313	20	0.003074458567967	0.0041112704228459	0.0025786167414383	11157.656652252
314	20	0.0031658388646642	0.0041110073813985	0.0025609107838247	11006.160277487
315	20	0.0031236938857172	0.0041583043560225	0.0024991401148327	11026.858537597
316	21	0.0031148355880284	0.0040399031316624	0.0025258848117396	11000.248656691
317	21	0.0031902515403898	0.0041092107131366	0.0025375692170203	11020.434653823
318	21	0.0030785784771139	0.0041583178912499	0.00252869282805	11080.106006285
319	21	0.0031936600353103	0.0041215247968568	0.0025670379502549	11014.394389769
320	21	0.0031418889094072	0.0042365782766834	0.0026131337045015	11149.597770217
321	21	0.0032133652585448	0.0041424262655923	0.0025391718268063	11033.569116022
322	21	0.0031923657323779	0.0040862384419955	0.002566201676933	11006.677483362
323	21	0.0031461817172963	0.0040965508890427	0.0025955447054018	11084.891930749
324	21	0.0031412631787244	0.0040619497189619	0.0025919923788546	11078.115802815
325	21	0.0031353183417118	0.0041466550425902	0.0025208888352253	11019.047333238
326	21	0.0031553994474389	0.0041514336285565	0.002542046905123	11018.129932487
327	21	0.003105979697521	0.0040742053485578	0.0025904643156048	11125.654302021
328	21	0.0031549853219017	0.0041079277407757	0.002538732002264	11009.782042909
329	21	0.0031096812516332	0.0041274312310149	0.0025682199613247	11095.570395498
330	21	0.0031519565562124	0.0041198505974704	0.0025532467962622	11012.873511793
331	22	0.0030909722632053	0.0041059197902589	0.0025425145010806	11074.343327915
332	22	0.0031678012333437	0.0041054851315381	0.0025130257864194	11022.370092252
333	22	0.0031258076244918	0.0040071737398262	0.0024998557107516	10994.581966274
334	22	0.0031308015654425	0.0041323590351105	0.0025869394276492	11098.906181285
335	22	0.0032179824168794	0.0040037206286287	0.002603622946573	11145.317885388
336	22	0.0030908575849891	0.0041129009380803	0.0025852550004909	11146.156477004
337	22	0.0031121792108983	0.0041439063425803	0.0025171381524825	11013.339928027
338	22	0.003119883505569	0.0040595275558572	0.0025907385251411	11104.107124719
339	22	0.0031371437029769	0.0039866893671909	0.0025037184373983	10991.793863993
340	22	0.0031009370045733	0.0040071730911486	0.0025413162734771	11037.43191633
341	22	0.0032768474389514	0.0040638794714857	0.002536843381402	11035.398323135
342	22	0.0031003800627149	0.0041120745103355	0.0026001338087746	11157.325096866
343	22	0.0031545336924927	0.0041010657772222	0.002578554249046	11046.983549398
344	22	0.0031389068732451	0.0040879989427131	0.0025654728289203	11043.321717767
345	22	0.0031367611365134	0.0041448704221098	0.0025500686129343	11033.136856532

346	23	0.003144170427076	0.0040405189634146	0.0025105814043146	11002.643117038
347	23	0.0031208759855442	0.0041199634752177	0.0025348711630061	11024.086675085
348	23	0.0030583488226862	0.0041179735750572	0.0025071691539436	11064.468095316
349	23	0.0031034428976233	0.0040107439105456	0.0025050226741636	10986.926217527
350	23	0.0031414506561016	0.0039774630981982	0.0025657289290502	11089.321934123
351	23	0.0031564453048978	0.0041823420769875	0.0024695579837203	11054.231083309
352	23	0.0031786111811131	0.004203716301549	0.0025211541858295	11043.486142561
353	23	0.0031051661047305	0.0039687722111949	0.0025559459847894	11072.330964077
354	23	0.0030632132867812	0.0041855990817783	0.002475213024111	11025.993713046
355	23	0.0031121901704425	0.0040319285217203	0.0025278589673036	11005.353923172
356	23	0.0030753333816585	0.004072843516708	0.0025157616008613	11044.997222963
357	23	0.0031393235181843	0.0041317524180868	0.0025691415570144	11058.213039131
358	23	0.0031790634521013	0.0039370514258901	0.0024923749303104	11023.187367198
359	23	0.0031924437994003	0.0040194646498511	0.0024653596554951	11031.250761202
360	23	0.0031529072495245	0.0039321607318536	0.0024814644348328	11003.239159094

B.2.3 SQP after SAE

	Label	Iter	Annot	Section1	Section2	Section3	Cost
1	1	1	START	0.00310344	0.00401074	0.00250502	10986.925632842
5	2	1	LINE	0.0029426421233814	0.0038881110652061	0.0027325767440789	11556.247565386
6	3	1	LINE	0.0030855520235787	0.0039970981314611	0.0025303345735391	11038.404071115
7	4	1	LINE	0.0031011110869627	0.0040089639061174	0.002508315813845	10984.565484706
8	5	2	NEWITER	0.0031011110869627	0.0040089639061174	0.002508315813845	10984.565484706
12	6	2	LINE	0.0028146825655394	0.0037905255334729	0.0025063966925447	11372.998971162
13	7	2	LINE	0.0030650805522989	0.0039814860162817	0.0025080744029471	11027.069391362
14	8	2	LINE	0.0030960897059285	0.0040051344610046	0.0025082821697119	10989.416740026
15	9	2	LINE	0.0031003061931675	0.0040083500716784	0.0025083104209155	10984.377053174
16	10	3	NEWITER	0.0031003061931675	0.0040083500716784	0.0025083104209155	10984.377053174
20	11	3	LINE	0.00300339732709	0.0037279959958529	0.0024969183120497	11226.270664326
21	12	3	LINE	0.0030858887198419	0.0039666408083928	0.0025066155767558	10992.451130067
22	13	3	LINE	0.0030955935806262	0.0039947166427961	0.0025077564303748	10986.991659741
23	14	3	LINE	0.003098760812247	0.004003879336881	0.0025081287538669	10985.231830023
24	15	3	LINE	0.0030997988921556	0.0040068824669883	0.0025082507852125	10984.657368974
25	16	3	LINE	0.0031001396045247	0.0040078681363623	0.0025082908376097	10984.469073617
26	17	3	LINE	0.003100251482237	0.0040081917948047	0.0025083039893793	10984.407271201
27	18	3	LINE	0.0031002882243741	0.0040082980885661	0.0025083083085964	10984.386977374
28	19	3	LINE	0.0031003002915777	0.0040083329985786	0.0025083097271549	10984.380312596
29	20	3	LINE	0.0031003042548682	0.004008344464244	0.002508310193059	10984.378123685
30	21	3	LINE	0.00300339732709	0.0037279959958529	0.0024969183120497	11226.270664326
31	22	3	LINE	0.0030906153065597	0.0039803146640958	0.0025071712100289	10989.779516947
32	23	3	LINE	0.0030993371045067	0.0040055465309201	0.0025081964998268	10984.912774636
33	24	4	NEWITER	0.0030993371045067	0.0040055465309201	0.0025081964998268	10984.912774636
37	25	4	LINE	0.0030972379245619	0.0036641145606244	0.0025068816279233	11350.696436905
38	26	4	LINE	0.0030991271865122	0.0039714033338905	0.0025080650126365	10989.042786229
39	27	4	LINE	0.003099269262165	0.003994511962652	0.0025081540051426	10982.559714878
40	28	5	NEWITER	0.003099269262165	0.003994511962652	0.0025081540051426	10982.559714878
44	29	5	LINE	0.0030911527605721	0.0022773909799844	0.0025018054080934	14428.508239162
45	30	5	LINE	0.0030984576120057	0.0038227998643852	0.0025075191454376	11155.409959872
46	31	5	LINE	0.003099188097149	0.0039773407528253	0.0025080905191721	10982.721041157
47	32	5	LINE	0.003099230334976	0.0039862765557753	0.0025081235569212	10980.789622866
48	33	6	NEWITER	0.003099230334976	0.0039862765557753	0.0025081235569212	10980.789622866
52	34	6	LINE	0.0030880042605093	0.0015	0.0024986832444542	18892.640724619
53	35	6	LINE	0.0030981077275293	0.0037376489001978	0.0025071795256745	11257.853847753
54	36	6	LINE	0.0030991180742313	0.0039614137902176	0.0025080291537965	10999.768742746
55	37	6	LINE	0.0030992176915157	0.0039834763660131	0.0025081129246903	10980.186491472
56	38	7	NEWITER	0.0030992176915157	0.0039834763660131	0.0025081129246903	10980.186491472
60	39	7	LINE	0.0030877447506265	0.0015	0.0024983100000702	18891.953560988
61	40	7	LINE	0.0030980703974268	0.0037351287294117	0.0025071326322283	11260.901990999
62	41	7	LINE	0.0030991029621068	0.0039586416023529	0.0025080148954441	11002.74623087
63	42	7	LINE	0.0030992062185748	0.003980992889647	0.0025081031217657	10979.6513205
64	43	8	NEWITER	0.0030992062185748	0.003980992889647	0.0025081031217657	10979.6513205
68	44	8	LINE	0.0030853110479826	0.0015	0.0024977157026903	18890.290785665
69	45	8	LINE	0.0030978167015156	0.0037328936006823	0.0025070643798582	11263.503385987
70	46	8	LINE	0.0030990672668689	0.0039561829607506	0.002507999247575	11005.379943271
71	47	8	LINE	0.0030991923234042	0.0039785118967574	0.0025080927343466	10981.469958535
72	48	8	LINE	0.0030992045848489	0.0039807011866759	0.0025081019004639	10979.588737861
73	49	9	NEWITER	0.0030992045848489	0.0039807011866759	0.0025081019004639	10979.588737861
77	50	9	LINE	0.0030850581706105	0.0015	0.0024976736677963	18890.150453921
78	51	9	LINE	0.003097789943425	0.0037326310680083	0.0025070590771972	11263.814518632
79	52	9	LINE	0.0030990631207065	0.0039558941748091	0.0025079976181373	11005.689909681
80	53	9	LINE	0.0030991904384346	0.0039782204854892	0.0025080914722313	10981.779773041
81	54	9	LINE	0.0030992031440275	0.0039804485256776	0.0025081008383418	10979.534559632
82	55	10	NEWITER	0.0030992031440275	0.0039804485256776	0.0025081008383418	10979.534559632

B.3 Monte-Carlo simulation

	Section1	Section2	Section3	Stress_beam28
1	0.0045663308856927	0.0029498492171963	0.0019121861233716	1.2226499031188E8
2	0.0046692907762321	0.0029360031751839	0.002092560755287	1.1176195248757E8
3	0.0046648735512399	0.002818422599263	0.0018209999765142	1.2836107550659E8
4	0.0046134616790455	0.0029706120628772	0.0019907433384046	1.1745769365586E8
5	0.0046250809295266	0.0030082683330667	0.0019503022467234	1.1988421851236E8
6	0.0045523066075103	0.0030601857200046	0.0019816297601764	1.1799687563887E8

7	0.0046627262391023	0.0030910746690982	0.0019372598820977	1.2068834782207E8
8	0.0046460650909472	0.0030392679504026	0.0019103367685679	1.2238263185557E8
9	0.0046693518455526	0.0028720712776768	0.0019634000623117	1.1908635505067E8
10	0.0045095184417238	0.0028236099332275	0.0020874450594548	1.1203565809057E8
11	0.004672805530112	0.0030710247158521	0.002016146466105	1.1598309906598E8
12	0.0044845141340081	0.0030518711797196	0.0019973943347102	1.170695688546E8
13	0.0046437413051991	0.0031126972422655	0.0017596428096774	1.3282020766893E8
14	0.0045526730296441	0.0032167791589486	0.0020169335587439	1.1593969032241E8
15	0.004610018819424	0.0032842567862996	0.0020574691266479	1.1366340139211E8
16	0.0046476344738361	0.0031220896285288	0.0017998936307529	1.2986203988009E8
17	0.0047097234403487	0.0032978032528677	0.00202811920622	1.1530155395607E8
18	0.0046221297232921	0.0030250001996253	0.0019991365476021	1.1696655691745E8
19	0.0045853532802319	0.0029267072448951	0.0020280515031101	1.1530475985186E8
20	0.004510817933606	0.0031000739599817	0.0017822030008653	1.3114704226578E8
21	0.0045878232044879	0.0030558936906698	0.001934687345409	1.2084871962399E8
22	0.0046491058860518	0.0030002359747997	0.001797400937851	1.3004074932504E8
23	0.0045466592984873	0.0030682348021355	0.0019071263054554	1.225889204033E8
24	0.0046644232965529	0.0030213487170601	0.0020272681533987	1.1534894529394E8
...	...	...	...	...
494	0.0046903010015453	0.0030491178620547	0.0019585390475546	1.1938155211932E8
495	0.0045404549747006	0.0029595526624506	0.0020026661231019	1.1676156160192E8
496	0.0045496374047243	0.0030954752725733	0.002059024440556	1.135775675159E8
497	0.0046678679671919	0.0028768456550916	0.001960770162691	1.1924552326994E8
498	0.0046607559055065	0.0028961403423457	0.0021336334209994	1.0961778631284E8
499	0.0046475659328288	0.003223577253519	0.0018890341945134	1.2375825926352E8
500	0.0045351078711727	0.0032004524837807	0.0020052851704349	1.1661079326272E8

Appendix C

Proof of Concept Code Listings

C.1 Analysis pattern

C.1.1 Example 1 WSDL file

Listing C.1: Example 1 WSDL file

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://formula" xmlns:
   apachesoap="http://xml.apache.org/xml-soap" xmlns:impl
   ="http://formula" xmlns:intf="http://formula" xmlns:
   wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsdlsoap
   ="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:xsd="
   http://www.w3.org/2001/XMLSchema">
<!--WSDL created by Apache Axis version: 1.4
Built on Apr 22, 2006 (06:55:48 PDT)-->
 <wsdl:types>
  <schema elementFormDefault="qualified" targetNamespace=
     "http://formula" xmlns="http://www.w3.org/2001/
     XMLSchema">
   <element name="squareValue">
    <complexType>
     <sequence>
      <element name="value" type="xsd:double"/>
     </sequence>
    </complexType>
   </element>
   <element name="squareValueResponse">
    <complexType>
```

```
16      <sequence>
17       <element name="squareValueReturn" type="xsd:double"
            />
18      </sequence>
19     </complexType>
20    </element>
21   </schema>
22  </wsdl:types>
23
24    <wsdl:message name="squareValueRequest">
25
26       <wsdl:part element="impl:squareValue" name="
            parameters"/>
27
28    </wsdl:message>
29
30    <wsdl:message name="squareValueResponse">
31
32       <wsdl:part element="impl:squareValueResponse" name=
            "parameters"/>
33
34    </wsdl:message>
35
36    <wsdl:portType name="Formula">
37
38       <wsdl:operation name="squareValue">
39
40          <wsdl:input message="impl:squareValueRequest"
               name="squareValueRequest"/>
41
42          <wsdl:output message="impl:squareValueResponse"
               name="squareValueResponse"/>
43
44       </wsdl:operation>
45
46    </wsdl:portType>
47
48    <wsdl:binding name="FormulaSoapBinding" type="impl:
         Formula">
49
50       <wsdlsoap:binding style="document" transport="http
            ://schemas.xmlsoap.org/soap/http"/>
51
```

```
52        <wsdl:operation name="squareValue">
53
54           <wsdlsoap:operation soapAction=""/>
55
56           <wsdl:input name="squareValueRequest">
57
58              <wsdlsoap:body use="literal"/>
59
60           </wsdl:input>
61
62           <wsdl:output name="squareValueResponse">
63
64              <wsdlsoap:body use="literal"/>
65
66           </wsdl:output>
67
68        </wsdl:operation>
69
70     </wsdl:binding>
71
72     <wsdl:service name="FormulaService">
73
74        <wsdl:port binding="impl:FormulaSoapBinding" name="
             Formula">
75
76           <wsdlsoap:address location="http://localhost
                :8080/Analysis/services/Formula"/>
77
78        </wsdl:port>
79
80     </wsdl:service>
81
82 </wsdl:definitions>
```

C.1.2 Example 2 WSDL file

Listing C.2: Example 2 WSDL file

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/
     wsdl/soap/" xmlns:tm="http://microsoft.com/wsdl/mime/
     textMatching/" xmlns:soapenc="http://schemas.xmlsoap.
```

```
   org/soap/encoding/" xmlns:mime="http://schemas.xmlsoap
   .org/wsdl/mime/" xmlns:tns="http://tempuri.org/
   Rosebrock" xmlns:s="http://www.w3.org/2001/XMLSchema"
   xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
    xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
   targetNamespace="http://tempuri.org/Rosebrock" xmlns:
   wsdl="http://schemas.xmlsoap.org/wsdl/">
3  <wsdl:types>
4    <s:schema elementFormDefault="qualified"
        targetNamespace="http://tempuri.org/Rosebrock">
5      <s:element name="Rosenbrock">
6        <s:complexType>
7          <s:sequence>
8            <s:element minOccurs="0" maxOccurs="1" name="
                inputValues" type="tns:ArrayOfDouble" />
9          </s:sequence>
10       </s:complexType>
11     </s:element>
12     <s:complexType name="ArrayOfDouble">
13       <s:sequence>
14         <s:element minOccurs="0" maxOccurs="unbounded"
              name="double" type="s:double" />
15       </s:sequence>
16     </s:complexType>
17     <s:element name="RosenbrockResponse">
18       <s:complexType>
19         <s:sequence>
20           <s:element minOccurs="1" maxOccurs="1" name="
                RosenbrockResult" type="s:double" />
21         </s:sequence>
22       </s:complexType>
23     </s:element>
24   </s:schema>
25 </wsdl:types>
26 <wsdl:message name="RosenbrockSoapIn">
27   <wsdl:part name="parameters" element="tns:Rosenbrock"
        />
28 </wsdl:message>
29 <wsdl:message name="RosenbrockSoapOut">
30   <wsdl:part name="parameters" element="tns:
        RosenbrockResponse" />
31 </wsdl:message>
32 <wsdl:portType name="ServiceSoap">
```

```
33      <wsdl:operation name="Rosenbrock">
34        <wsdl:input message="tns:RosenbrockSoapIn" />
35        <wsdl:output message="tns:RosenbrockSoapOut" />
36      </wsdl:operation>
37    </wsdl:portType>
38    <wsdl:binding name="ServiceSoap" type="tns:ServiceSoap"
         >
39      <soap:binding transport="http://schemas.xmlsoap.org/
           soap/http" />
40      <wsdl:operation name="Rosenbrock">
41        <soap:operation soapAction="http://tempuri.org/
             Rosebrock/Rosenbrock" style="document" />
42        <wsdl:input>
43          <soap:body use="literal" />
44        </wsdl:input>
45        <wsdl:output>
46          <soap:body use="literal" />
47        </wsdl:output>
48      </wsdl:operation>
49    </wsdl:binding>
50    <wsdl:binding name="ServiceSoap12" type="tns:
         ServiceSoap">
51      <soap12:binding transport="http://schemas.xmlsoap.org
           /soap/http" />
52      <wsdl:operation name="Rosenbrock">
53        <soap12:operation soapAction="http://tempuri.org/
             Rosebrock/Rosenbrock" style="document" />
54        <wsdl:input>
55          <soap12:body use="literal" />
56        </wsdl:input>
57        <wsdl:output>
58          <soap12:body use="literal" />
59        </wsdl:output>
60      </wsdl:operation>
61    </wsdl:binding>
62    <wsdl:service name="Service">
63      <wsdl:port name="ServiceSoap" binding="tns:
           ServiceSoap">
64        <soap:address location="http://localhost:1430/
             Rosenbrock/Service.asmx" />
65      </wsdl:port>
66      <wsdl:port name="ServiceSoap12" binding="tns:
           ServiceSoap12">
```

```
67        <soap12:address location="http://localhost:1430/
              Rosenbrock/Service.asmx" />
68      </wsdl:port>
69    </wsdl:service>
70  </wsdl:definitions>
```

C.2 Experiment loop pattern

Listing C.3: Random DOE WSDL file

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <wsdl:definitions targetNamespace="http://random" xmlns:
       apachesoap="http://xml.apache.org/xml-soap" xmlns:impl
       ="http://random" xmlns:intf="http://random" xmlns:wsdl
       ="http://schemas.xmlsoap.org/wsdl/" xmlns:wsdlsoap="
       http://schemas.xmlsoap.org/wsdl/soap/" xmlns:xsd="http
       ://www.w3.org/2001/XMLSchema">
3  <!--WSDL created by Apache Axis version: 1.4
4  Built on Apr 22, 2006 (06:55:48 PDT)-->
5   <wsdl:types>
6    <schema elementFormDefault="qualified" targetNamespace=
         "http://random" xmlns="http://www.w3.org/2001/
         XMLSchema">
7     <element name="generateDOE">
8      <complexType>
9       <sequence>
10       <element name="num" type="xsd:int"/>
11      </sequence>
12     </complexType>
13    </element>
14    <element name="generateDOEResponse">
15     <complexType>
16      <sequence>
17       <element maxOccurs="unbounded" name="
             generateDOEReturn" type="impl:ArrayOf_xsd_double
             "/>
18      </sequence>
19     </complexType>
20    </element>
21    <complexType name="ArrayOf_xsd_double">
22     <sequence>
```

```
23      <element maxOccurs="unbounded" minOccurs="0" name="
           item" type="xsd:double"/>
24     </sequence>
25    </complexType>
26   </schema>
27  </wsdl:types>
28
29    <wsdl:message name="generateDOERequest">
30
31       <wsdl:part element="impl:generateDOE" name="
           parameters"/>
32
33    </wsdl:message>
34
35    <wsdl:message name="generateDOEResponse">
36
37       <wsdl:part element="impl:generateDOEResponse" name=
           "parameters"/>
38
39    </wsdl:message>
40
41    <wsdl:portType name="RandomDOE3">
42
43       <wsdl:operation name="generateDOE">
44
45          <wsdl:input message="impl:generateDOERequest"
              name="generateDOERequest"/>
46
47          <wsdl:output message="impl:generateDOEResponse"
              name="generateDOEResponse"/>
48
49       </wsdl:operation>
50
51    </wsdl:portType>
52
53    <wsdl:binding name="RandomDOE3SoapBinding" type="impl:
       RandomDOE3">
54
55       <wsdlsoap:binding style="document" transport="http
          ://schemas.xmlsoap.org/soap/http"/>
56
57       <wsdl:operation name="generateDOE">
58
```

```
59          <wsdlsoap:operation soapAction=""/>
60
61          <wsdl:input name="generateDOERequest">
62
63             <wsdlsoap:body use="literal"/>
64
65          </wsdl:input>
66
67          <wsdl:output name="generateDOEResponse">
68
69             <wsdlsoap:body use="literal"/>
70
71          </wsdl:output>
72
73       </wsdl:operation>
74
75    </wsdl:binding>
76
77    <wsdl:service name="RandomDOE3Service">
78
79       <wsdl:port binding="impl:RandomDOE3SoapBinding"
            name="RandomDOE3">
80
81          <wsdlsoap:address location="http://localhost
               :8080/Analysis/services/RandomDOE3"/>
82
83       </wsdl:port>
84
85    </wsdl:service>
86
87 </wsdl:definitions>
```

C.3 Optimization loop pattern

Listing C.4: Optimization WSDL file

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/
     wsdl/soap/" xmlns:tm="http://microsoft.com/wsdl/mime/
     textMatching/" xmlns:soapenc="http://schemas.xmlsoap.
     org/soap/encoding/" xmlns:mime="http://schemas.xmlsoap
     .org/wsdl/mime/" xmlns:tns="http://tempuri.org/L-BSFG"
```

```
   xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:
  soap12="http://schemas.xmlsoap.org/wsdl/soap12/" xmlns
  :http="http://schemas.xmlsoap.org/wsdl/http/"
  targetNamespace="http://tempuri.org/L-BSFG" xmlns:wsdl
  ="http://schemas.xmlsoap.org/wsdl/">
<wsdl:types>
  <s:schema elementFormDefault="qualified"
     targetNamespace="http://tempuri.org/L-BSFG">
    <s:element name="HelloWorld">
      <s:complexType />
    </s:element>
    <s:element name="HelloWorldResponse">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="
             HelloWorldResult" type="s:string" />
        </s:sequence>
      </s:complexType>
    </s:element>
    <s:element name="init">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="
             startValues" type="tns:ArrayOfInt" />
        </s:sequence>
      </s:complexType>
    </s:element>
    <s:complexType name="ArrayOfInt">
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="unbounded"
           name="int" type="s:int" />
      </s:sequence>
    </s:complexType>
    <s:element name="initResponse">
      <s:complexType />
    </s:element>
    <s:element name="hasConverged">
      <s:complexType />
    </s:element>
    <s:element name="hasConvergedResponse">
      <s:complexType>
        <s:sequence>
```

```
36            <s:element minOccurs="1" maxOccurs="1" name="
                 hasConvergedResult" type="s:boolean" />
37          </s:sequence>
38        </s:complexType>
39      </s:element>
40      <s:element name="iterate">
41        <s:complexType>
42          <s:sequence>
43            <s:element minOccurs="0" maxOccurs="1" name="
                 functionValues" type="tns:ArrayOfDouble"
                 />
44          </s:sequence>
45        </s:complexType>
46      </s:element>
47      <s:complexType name="ArrayOfDouble">
48        <s:sequence>
49          <s:element minOccurs="0" maxOccurs="unbounded"
               name="double" type="s:double" />
50        </s:sequence>
51      </s:complexType>
52      <s:element name="iterateResponse">
53        <s:complexType>
54          <s:sequence>
55            <s:element minOccurs="0" maxOccurs="1" name="
                 iterateResult" type="tns:
                 ArrayOfArrayOfDouble" />
56          </s:sequence>
57        </s:complexType>
58      </s:element>
59      <s:complexType name="ArrayOfArrayOfDouble">
60        <s:sequence>
61          <s:element minOccurs="0" maxOccurs="unbounded"
               name="ArrayOfDouble" nillable="true" type="
               tns:ArrayOfDouble" />
62        </s:sequence>
63      </s:complexType>
64    </s:schema>
65  </wsdl:types>
66  <wsdl:message name="HelloWorldSoapIn">
67    <wsdl:part name="parameters" element="tns:HelloWorld"
        />
68  </wsdl:message>
69  <wsdl:message name="HelloWorldSoapOut">
```

```
70      <wsdl:part name="parameters" element="tns:
            HelloWorldResponse" />
71    </wsdl:message>
72    <wsdl:message name="initSoapIn">
73      <wsdl:part name="parameters" element="tns:init" />
74    </wsdl:message>
75    <wsdl:message name="initSoapOut">
76      <wsdl:part name="parameters" element="tns:
            initResponse" />
77    </wsdl:message>
78    <wsdl:message name="hasConvergedSoapIn">
79      <wsdl:part name="parameters" element="tns:
            hasConverged" />
80    </wsdl:message>
81    <wsdl:message name="hasConvergedSoapOut">
82      <wsdl:part name="parameters" element="tns:
            hasConvergedResponse" />
83    </wsdl:message>
84    <wsdl:message name="iterateSoapIn">
85      <wsdl:part name="parameters" element="tns:iterate" />
86    </wsdl:message>
87    <wsdl:message name="iterateSoapOut">
88      <wsdl:part name="parameters" element="tns:
            iterateResponse" />
89    </wsdl:message>
90    <wsdl:portType name="ServiceSoap">
91      <wsdl:operation name="HelloWorld">
92        <wsdl:input message="tns:HelloWorldSoapIn" />
93        <wsdl:output message="tns:HelloWorldSoapOut" />
94      </wsdl:operation>
95      <wsdl:operation name="init">
96        <wsdl:input message="tns:initSoapIn" />
97        <wsdl:output message="tns:initSoapOut" />
98      </wsdl:operation>
99      <wsdl:operation name="hasConverged">
100       <wsdl:input message="tns:hasConvergedSoapIn" />
101       <wsdl:output message="tns:hasConvergedSoapOut" />
102     </wsdl:operation>
103     <wsdl:operation name="iterate">
104       <wsdl:input message="tns:iterateSoapIn" />
105       <wsdl:output message="tns:iterateSoapOut" />
106     </wsdl:operation>
107   </wsdl:portType>
```

```
108   <wsdl:binding name="ServiceSoap" type="tns:ServiceSoap"
          >
109     <soap:binding transport="http://schemas.xmlsoap.org/
            soap/http" />
110     <wsdl:operation name="HelloWorld">
111       <soap:operation soapAction="http://tempuri.org/L-
            BSFG/HelloWorld" style="document" />
112       <wsdl:input>
113         <soap:body use="literal" />
114       </wsdl:input>
115       <wsdl:output>
116         <soap:body use="literal" />
117       </wsdl:output>
118     </wsdl:operation>
119     <wsdl:operation name="init">
120       <soap:operation soapAction="http://tempuri.org/L-
            BSFG/init" style="document" />
121       <wsdl:input>
122         <soap:body use="literal" />
123       </wsdl:input>
124       <wsdl:output>
125         <soap:body use="literal" />
126       </wsdl:output>
127     </wsdl:operation>
128     <wsdl:operation name="hasConverged">
129       <soap:operation soapAction="http://tempuri.org/L-
            BSFG/hasConverged" style="document" />
130       <wsdl:input>
131         <soap:body use="literal" />
132       </wsdl:input>
133       <wsdl:output>
134         <soap:body use="literal" />
135       </wsdl:output>
136     </wsdl:operation>
137     <wsdl:operation name="iterate">
138       <soap:operation soapAction="http://tempuri.org/L-
            BSFG/iterate" style="document" />
139       <wsdl:input>
140         <soap:body use="literal" />
141       </wsdl:input>
142       <wsdl:output>
143         <soap:body use="literal" />
144       </wsdl:output>
```

```
145     </wsdl:operation>
146   </wsdl:binding>
147   <wsdl:binding name="ServiceSoap12" type="tns:
         ServiceSoap">
148     <soap12:binding transport="http://schemas.xmlsoap.org
         /soap/http" />
149     <wsdl:operation name="HelloWorld">
150       <soap12:operation soapAction="http://tempuri.org/L-
           BSFG/HelloWorld" style="document" />
151       <wsdl:input>
152         <soap12:body use="literal" />
153       </wsdl:input>
154       <wsdl:output>
155         <soap12:body use="literal" />
156       </wsdl:output>
157     </wsdl:operation>
158     <wsdl:operation name="init">
159       <soap12:operation soapAction="http://tempuri.org/L-
           BSFG/init" style="document" />
160       <wsdl:input>
161         <soap12:body use="literal" />
162       </wsdl:input>
163       <wsdl:output>
164         <soap12:body use="literal" />
165       </wsdl:output>
166     </wsdl:operation>
167     <wsdl:operation name="hasConverged">
168       <soap12:operation soapAction="http://tempuri.org/L-
           BSFG/hasConverged" style="document" />
169       <wsdl:input>
170         <soap12:body use="literal" />
171       </wsdl:input>
172       <wsdl:output>
173         <soap12:body use="literal" />
174       </wsdl:output>
175     </wsdl:operation>
176     <wsdl:operation name="iterate">
177       <soap12:operation soapAction="http://tempuri.org/L-
           BSFG/iterate" style="document" />
178       <wsdl:input>
179         <soap12:body use="literal" />
180       </wsdl:input>
181       <wsdl:output>
```

```
182         <soap12:body use="literal" />
183       </wsdl:output>
184     </wsdl:operation>
185   </wsdl:binding>
186   <wsdl:service name="Service">
187     <wsdl:port name="ServiceSoap" binding="tns:
          ServiceSoap">
188       <soap:address location="http://localhost:3572/L-
            BGFS/Service.asmx" />
189     </wsdl:port>
190     <wsdl:port name="ServiceSoap12" binding="tns:
          ServiceSoap12">
191       <soap12:address location="http://localhost:3572/L-
            BGFS/Service.asmx" />
192     </wsdl:port>
193   </wsdl:service>
194 </wsdl:definitions>
```

Bibliography

[AA07] Alexandre Alves and Assaf Arkin. Web services business process execution language version 2.0. Technical report, Oasis, 2007.

[Bey98] Hans-Georg Beyer. *The Theory of Evolution Strategies*. Springer, 1998.

[BM04] P. V. Biron and A. Malhotra. W3c recommendation, xml schema part 2: Datatypes second edition. `http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/`, 2004.

[Boc00] Sergey Bochkanov. L-bfgs algorithm for multivariate optimization (bsd licensed). `http://www.alglib.net/optimization/lbfgs.php`, 2000.

[CCMW01] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. W3c note, web services definition language (wsdl) 1.1. `http://www.w3.org/TR/2001/NOTE-wsdl-20010315`, 2001.

[CD99] J. Clarck and S. DeRose. W3c recommandation, xml path language (xpath) version 1.0. `http://www.w3.org/TR/1999/RECxpath-19991116`, 1999.

[CNdPJ02] McCulloch F Colin, Tzannetakis Nick, and Van de Peer Joost. Multi-disciplinary design optimization in support of the functional performance engineering process. Technical report, LMS International, 2002.

[dPJ02] Van de Peer Joost. Design by objective: Revolutionizing product development with process integration and design optimization. Technical report, Noesis Solutions, 2002.

[End09] Active Endpoints. Activevos tutorial. `http://infocenter.activevos.com/infocenter/ActiveVOS/v60/index.jsp?topic=/com.activee.bpel.doc/html/UG3.html`, 2009.

[Esp08] Dino Esposito. *Programming Microsoft ASP.NET 3.5*. Microsoft, 2008.

[GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[GJSB00] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *The Java Language Specification: Second Edition*. Addison-Wesley, 2000.

[Hof03] A. H. M. Ter Hofstede. Yawl: Yet another workflow language (revised version. Technical report, 2003.

[JDVJ08] Niels Joncheere, Dirk Deridder, Ragnhild Van Der Straeten, and Viviane Jonckers. A framework for advanced modularization and data flow in workflow systems. In *Proceedings of the 6th International Conference on Service Oriented Computing (ICSOC 2008)*, volume 5364 of *Lecture Notes in Computer Science*, pages 592–598, Sydney, NSW, Australia, December 2008. Springer.

[Kne04] Sean Kneipp. Yawl engine user manual. Technical report, Faculty of Information Technology, 2004.

[LA93] Thomas J. Lorenzen and Virgil L. Anderson. *Design of Experiments, A No-Name Approach*. Marcel Dekker, Inc., 1993.

[LKD+08] Robert Lorenz, Lars M. Kristensen, Philippe Darondeau, Robin Bergenthum, and Sebastian Mauser. *Applications and Theory of Petri Nets*. Springer Berlin / Heidelberg, 2008.

[Min07] Nicholas Charles Russell BSc MinfTech. Foundations of process-aware information systems. Technical report, 2007.

[MMAC08] Raymond H. Myers, Douglas C. Montgomery, and Christine M. Anderson-Cook. *Response Surface Methodology: Process and Product Optimization Using Designed Experiments*. Wiley, John & Sons, Incorporated, 2008.

[NdPJS03] Tzannetakis Nick, Van de Peer Joost, and Donders Stijn. A system approach to design for six sigma through best in class simulation process integration & design optimization. Technical report, LMS International, 2003.

[P.93] Spellucci P. *Numerische Verfahren der nichtlinearen Optimierung*. Birkhäuser, 1993.

[Pla03] David S. Platt. *Introducing Microsoft .NET, Third Edition*. Microsoft, 2003.

[PR09] Srinath Perera and Ajith Ranabahu. Axis2 - the future of web services. `http://www.jaxmag.com/itr/online\_artikel/psecom,id,747,nodeid,147.html`, 2009.

[Sol08] Noesis Solutions. Optimus 8. `http://www.noesissolutions.com`, 2008.

[Sto85] J. Stoer. Foundations of recursive quadratic programming methods for solving nonlinear programs. *Computational Mathematical Programming*, 15, 1985.

[TBMM04] H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. W3c recommendation, xml schema part 1: Structures second edition. `http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/`, 2004.

[TOH+99] Peri Tarr, Harold Ossher, William Harrison, Stanley M. Sutton, and Jr. N degrees of separation: Multi-dimensional separation of concerns. pages 107–119, 1999.

[vdABtH+00] W.M.P. van der Aalst, A.P. Barros, A.H.M. ter Hofstede, B. Kiepuszewski, and B. Advanced workflow patterns, 2000.

[vdAH03] W.M.P. van der Aalst and A. H. M. Ter Hofstede. Yawl: Yet another workflow language. *Information Systems*, 30:245–275, 2003.

[vdAHW03] W.M.P. van der Aalst, A. H. M. Ter Hofstede, and M. Weske. Business process management: A survey. In *Proceedings of the 1st International Conference on Business Process Management, volume 2678 of LNCS*, pages 1–12. Springer-Verlag, 2003.

[vdAtH09] W.M.P. van der Aalst and A.H.M. ter Hofstede. Workflow patterns initiative. `http://www.workflowpatterns.com/about/index.php`, 2009.

[vdAtHKB02] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow patterns. Technical report, Queensland University of Technology, 2002.