4 METHODOLOGY

4.1 Research Aims - Revisited

Having proposed a novel object-based representation for evolutionary search and exploration of early lifecycle software designs, it is perhaps useful at this point to briefly revisit the research aims of this thesis in order to explain the methodological issues associated with their investigation. In essence, this thesis hypothesises that computationally intelligent tool support can greatly assist the designer with the complexities of early lifecycle software design. Specifically, it is hypothesised that interactive evolutionary computing affords significant opportunities for combined quantitative and qualitative multi-objective search and exploration. It is also hypothesised that software agents offer great potential for the facilitation of not only collaborative two-way interaction, but also machine learning techniques which may reveal new knowledge relating to the nature of software design elegance, whilst enhancing designer interaction by reducing user fatigue. Thus investigations will address the following research areas:

- How to represent the design problem and early lifecycle design solution such that they are sufficiently abstract for human comprehension but also effective for computational evolutionary search.
- How to effectively explore and exploit the software design solution search space using evolutionary computing to arrive at useful and innovative UML class software designs.
- How to facilitate collaborative designer / computer interaction using autonomous software agents.
- How software design elegance can be measured quantitatively.
- How machine-based quantitative fitness metrics can be effectively integrated with qualitative designer evaluations in multi-objective evolutionary search.
- How to reduce interaction fatigue by machine learning techniques such as dynamic interaction and reward-based learning.

4.2 Organization of Investigation

It quickly becomes apparent that there are too many research aims to be investigated in a single enquiry. Rather, it is necessary to adopt an iterative approach whereby individual research aims are investigated and subsequent results are evaluated before

proceeding to the next experiment. Of course, there are many dependencies between the research aims. However, pragmatically, it is necessary to break down experimentation into specific investigations that are aligned to the research aims, and conduct the investigations sequentially such that any successes and failures from one investigation are exploited in subsequent ones. Accordingly, investigations have been organised into four broad phases as follows:

- *Local Search:* Firstly, an initial set of experiments will trial the performance of the novel object-based representation and associated genetic operators using manual parameter 'tuning'. Secondly, experiments will be conducted to build upon the initial findings and investigate how dynamic parameter control might more effectively explore and exploit the software design solution search space.
- *Global Search:* Experiments will be conducted into global multi-objective search as the basis for effective narrowing of the global search space into local searches of promising zones of useful and interesting class designs. This narrowing of the global search space to local search(es) is facilitated by software agents as part of an interactive early lifecycle software design episode. Taken together, the software agents constitute an interactive design framework for collaborative designer / computer interaction.
- *Empirical Investigation:* Observations will be conducted of software designers working with the interactive design framework in an industrial early lifecycle software design situation.
- *Qualitative Elegance:* Experiments will be conducted to explore ways to best capture human software design elegance intentions. Further experiments will then be conducted to effectively combine qualitative evaluation of software design elegance with dynamically controlled quantitative local search. As part of these experiments, trials will be carried out to investigate the reduction of interactive user fatigue.

4.3 Strategy

The overall strategy applied to the investigations is driven by the designer-centred, interactive nature of the computational support framework. Thus effective exploration of the software design solution space, enabled by an interactive framework of software agents, is crucial. The interactive nature of the computational support framework is also

essential for designer qualitative evaluation of software design elegance. This investigative strategy is enumerated in more detail for each phase of investigation later in this chapter.

A further key factor in all investigation is the availability of plausible and representative test design problems for early lifecycle software design. According to a broad categorisation of Eiben and Smith (2003), there are three possible sources of test problems for evolutionary algorithms, namely:

- problem instances from academic or industrial benchmark repositories,
- problem instances created by a problem instance generator, and
- real-life problem instances.

Unfortunately, benchmark software design problems do not appear readily in either academic or industrial repositories. In academe, many books relating to teaching early lifecycle software design present small examples of design problems that are typically constrained in scope and complexity for teaching purposes. Such teaching example design problems are not likely to be representative of design problems encountered in the field. Moreover, in industry, it is interesting to note that while there exist benchmark evaluations for hardware processors and components (e.g. Intel, 2010), recognised benchmark software design problems are less readily available, possibly for reasons of commercial confidentiality. In the research arena, many problem instances have been generated to test evolutionary algorithms e.g. the De Jong test suite (1975), Deb (2001). However, equivalent software design problem instances seem not to have been generated in the research arena. Therefore, real-life software problem instances have been selected for use in the investigations of this thesis. Having been drawn from real-life, such software design problems are highly relevant to the application domain (of software design). Indeed, three real-life design problems have been selected to provide not only an appropriate range of problem domain but also a range of problem scale. Despite this, it is not entirely possible to precisely assess how representative the three design problems might be of the software design field as a whole, since there is no recognised repository of software design problems available for comparison. However, the second and third of the three design problems have been drawn from fully enterprise scale industrial software developments, and are decidedly non-trivial in size and complexity. Full specifications of the three real-world design problems used in experimental investigations are given later in this chapter.

Based upon the designer-centred, interactive investigation strategy nature of the computational support framework and driven by real-world software design problems, the following aspects of research methodology are described in the following sections for each phase of investigation:

- How is the data is to be arrived at?
- How is the data to be analysed?

Any strengths and weaknesses of the investigation approach are also identified. Ethical considerations are also addressed where appropriate.

4.3.1 Local Search

Research Aim:

- How to represent the design problem and early lifecycle design solution such that they are sufficiently abstract for human comprehension but also effective for computational evolutionary search.

Methodology:

- In the initial parameter 'tuning' experiments, investigations are conducted using the novel object-based representation and associated genetic operators. Evolutionary algorithm population size is investigated, as is selection, crossover and mutation mechanisms. The performance of software design average class cohesion and design coupling are compared as fitness functions. Consistent with the overall investigation strategy, algorithm speed in important and so will be measured. As search will be interactive in later experiments, exploration is also important and so it is essential to preserve a range of solutions so that the final evaluation decision can be made by the designer. It is usually neither possible nor desirable to prejudge designer decisions before they have seen examples drawn from a range of possible designs. Because of this, average population fitness and standard deviation are recorded as search progresses. As interactive search can be repetitive, good design solutions should be arrived at quickly rather than taking large amounts of time to find any 'global optimum' (which the research literature suggests is unlikely to exist). It is thus important that the local search is largely repeatable, and so each investigation comprises 50 runs of local search, enabling population average fitness and standard deviation to be calculated. As local search is intended to be designer interactive in later phases

of investigation, the termination condition in this investigation will be specific to this phase of testing i.e. search will run until it has been empirically determined that population convergence has occurred.

- After manual parameter tuning for a single design problem, the performance of fitness functions, selection mechanisms, diversity preservation mechanisms (crossover and mutation) will be analysed and compared. Average population fitness at convergence will also be compared with values obtained for a manually produced design. Speed, repeatability and diversity preservation of local search will be assessed by examination of the average population fitness curves and standard deviation based on 50 runs. Colourful, clear visualisations of class designs are also produced for later empirical validation.

Research Aim:

- How to effectively explore and exploit the software design solution search space using evolutionary computing to arrive at useful and innovative UML class software designs.

Methodology:

- There may be a weakness relating to parameter 'tuning'. Given the number of parameters to tune, and the many possible combinations thereof, it is difficult to establish definitively if manually tuned parameters are optimal. Thus a second investigation will assess the performance various dynamic parameter control mechanisms, including simulated annealing (e.g. Davis, 1987), the '1/5 success rule' (e.g. Rechenberg, 1973), and self-adaptation (e.g. Schwefel, 1995). Investigations of local search with dynamic parameter control will be at this point extended to further example design problems. As in initial experiments, average population fitness at convergence will be compared with values obtained for the three manually produced designs. Speed, repeatability and diversity preservation of local search will be also be assessed by examination of mutation probabilities, average population fitness curves with standard deviation based on 50 runs.

4.3.2 Global Search

Research Aim:

- How to facilitate collaborative designer / computer interaction using autonomous software agents.

Methodology:

- Investigations will be conducted into the effectiveness of multi-objective global search. The speed of evolutionary search is important and so will be measured. However, within an interactive search strategy, diversity preservation is also important. Diversity in the global search is measured by the number of 'fronts' of equivalent optimality in the population, as is the diversity within each front. Nevertheless, because of the discrete nature of the object-based representation, as population fitness increases, a corresponding loss of diversity may be evident, making the global search termination crucial. Thus as global search progresses, a software agent monitors the utility of the population with respect to the trade-off between increasing population fitness versus loss of diversity. As speed of global search is important, the utility of the fitness / diversity trade-off determines when global search is halted. Three halting tactics are trialled in the utility agent, and compared for termination of global search at
    - zero utility,
    - a single fall in utility, and
    - two contiguous falls in utility.
- Population utility is calculated over 50 runs for example design problems, and average population utility and standard deviation calculated are calculated for each halting tactic to determine the most effective for an interactive search. In this way, the global search space is effectively narrowed by the framework of agents to design zones for interactive local search.

4.3.3 Empirical investigation

Research Aims: this phase of investigation not only addresses all three previous research aims, but also sets out to evaluate the performance of interactive framework in an industrial situation to validate the framework empirically.

The method employed in the empirical investigation is to observe two early lifecycle software design episodes in an industrial setting. The first episode is conducted without the support of the computational framework (i.e. design is performed

manually); in the second episode the designers are supported by the interactive framework. The same participants take part in both episodes. Effectively, the manual episode constitutes a baseline (or control) against which comparisons and contrasts with the computational interactive framework supported episode may then be drawn. The industrial setting is an in-house Information Systems Department at the University of the West of England, UK. The design domain under observation is a development of the student administration system which relates to recording and tracking personal student development during their undergraduate studies. During both design episodes, the participating software designers are observed; both video and audio recordings are made. Following each episode, transcripts are prepared and analyzed with respect to the following data:

- *Design mode:* are the designers focussing on the problem or solution space? Is the thrust of their activity convergent or divergent? Is the medium of their activity verbal, sketching, writing, using the computational support framework?
- *Design activities:* are the designers primarily generating, evaluating, trading-off or scoping designs? Or are the designers perhaps reflecting in silence on designs?
- *Design events:* are the designers requesting clarification, explaining their understanding, suddenly discovering a design or realising a constraint or inferred requirement? Or are the designers inspecting a class diagram or adding a class diagram to a portfolio?

While these observational data give an indication of the richness and effectiveness of the software design episodes, there are limits to this method. For example, if there are small numbers of designers, how generalisable might be the results when the numbers of participants are few? Also, how representative is this sample of the larger population of software designers? These questions are addressed during analysis of the data.

With regard to the ethics of this observational study, a number of steps have been taken to ensure that the research participants are treated with care, sensitivity and respect for their status as human beings. For example, the concept of informed consent is applied to ensure that the participants are provided with an overview of the key aspects of the research. The methods of observation are explained to the participants, as are the types of data analysed. Details of the study are provided to the participants on an information sheet, which the participants are invited to sign to indicate their understanding of the study and to provide their informed consent. As part of the study,

anonymity and confidentiality are crucial. Thus roles played by individuals in the study e.g. “designer 1”, “designer 2” etc. are recorded rather than participant names. Any published results refer to participant roles rather than names. Participants are provided with copies of all published results. Electronic copies of recordings and raw transcripts are held in secure storage until destruction after an appropriate interval of time. The ethical approach of this observational study has been approved by the Research Committee of the Faculty of Environment and Technology at the University of the West of England prior to commencement of the study.

4.3.4 Qualitative Elegance

Research Aim:

- How software design elegance can be measured quantitatively.

Methodology:

- Four novel quantitative measures of software design elegance are formulated based upon software design symmetry and distribution with relation to the allocation of attributes and methods among classes. Within local search using design coupling as a fitness function for all three design problems, an elegance agent calculates the four elegance measures to obtain best, average and worst elegance values for each generation. The termination condition is once again specific to this phase of testing i.e. search will run until it has been empirically determined that population convergence has occurred with respect to design coupling.

Research aim:

- How machine-based quantitative fitness metrics can be effectively integrated with qualitative designer evaluations in multi-objective evolutionary search.

Methodology:

- Local search is conducted using design coupling and the four quantitative elegance measures as weighted fitness functions. At the beginning of local search, coupling is weighted at 100% while elegance measures are weighted at 0%. At appropriate interactive generational intervals, the elegance agent selects the best software design from the population based upon one of the four quantitative elegance measures chosen at random. The software designer is

presented with a colourful visual depiction of this software design. The designer is then asked to subjectively evaluate the design according to an ordinal star rating, where one star (‘*’) is poor and five stars (‘*****’) is good. The elegance agent receives the star rating as reward, and then calculates a rolling average reward based on this and previous rewards for the quantitative measure over the local search. The appropriate weighting of the chosen elegance measure is adjusted to reflect the rolling average and so designer preferred elegance measures are increasingly weighted as interactive search progresses. Therefore the four elegance measures, the rolling reward averages and the relative weights of design coupling are measured at each generation of search. As this search is interactive, termination occurs at any point at the request of the designer.

Research Aim:

- How to reduce interaction fatigue by machine learning techniques such as dynamic interaction and reward-based learning.

Methodology:

- A balance is required between the designer and the software agent interactive support framework to jointly and collaboratively steer the direction of multi-objective local search. To investigate this balance, the interactive interval (in terms of evolutionary generations) is varied dynamically depending on average coupling fitness of the population. At the beginning of search, average population design coupling is poor, and so the interactive interval will be high (in terms of generations) to reduce user fatigue. However, as average population design coupling improves, the interactive interval gradually decreases, to increasingly allow the designer’s qualitative elegance measures to increasingly influence to direction of local search. Therefore local search will be conducted over a range of dynamic interaction intervals for a given example design problem and the number of interactions recorded. Once again, as local search is interactive, termination occurs at any point at the request of the designer.

4.4 Example Software Design Problem Specifications

4.4.1 Cinema Booking System

The first example design problem used in this thesis is a generalized abstraction of a cinema booking system, derived from a number of established internet-based cinema booking systems existing in the UK. The example design problem domain addresses, for example, making an advance booking for a showing of a film, and payment for tickets on attending the cinema auditorium. A detailed specification of the use cases of the Cinema Booking System design problem is available in appendix G, and comprises 15 actions, 16 data and 39 uses. The author's manual early lifecycle UML class design is derived from the use case specification and shown in UML notation in figure 4.1. The direction of each 'external' use (i.e. couple) between classes is shown with a solid line and an arrowhead; for the sake of clarity, uses 'internal' to a class (i.e. cohesive uses) are not shown.

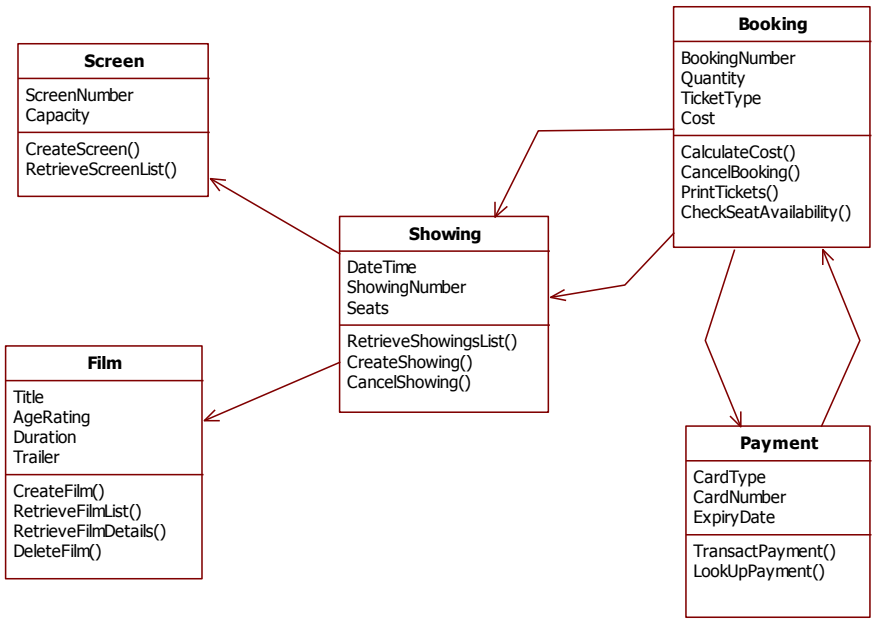

Figure 4.1. Cinema Booking System Manual Design.

Using the cohesion metric described previously in section 3.1, cohesion values of the manual design classes for the Cinema Booking System are shown in table 4.1. With regard to the class coupling metric, the authors' manual design in figure 4.1 has six 'external' uses. The total number of uses in the design problem specification is 39, thus the coupling value of the manual design is 6 / 39, or 0.154. Applying equation (1) from

section 3.1, the cardinality of the resulting design solution search space derived from this example problem domain is 2.77248 x 10^{19}.

Table 4.1. Cohesion Values of Manual Design for CBS

Class	Cohesion value
Film	0.75
Showing	0.2185
Screen	1.0
Booking	0.555
Payment	0.625
Average	0.62975

4.4.2 Graduate Development Program

The second example design problem domain is an extension to an undergraduate student administration system performed by the in-house Information Systems Department at the University of the West of England, UK. Over recent years, the University of the West of England University has sought to record and manage outcomes relating to personal student development during their studies. A strategic decision was made to extend the capabilities of the existing student administration system to be able to record and track student's personal development in parallel with their academic achievement. As part of this extension, rules for validation of completion of personal development outcomes are required, as is the ability to extract various reports. The extension was designed, implemented and deployed into a live environment in 2008. A detailed specification of the use cases used in the development is available in appendix H, and comprises 12 action, 43 data and 121 uses. Adhering to the use case specification, the author's manual class design is shown in UML notation in figure 4.2. Using the cohesion metric described in the previous section, cohesion values of the manual design classes for the Graduate Development Program are shown in table 4.2.

Although the author's manual design in figure 4.2 reveals a larger scale software design than the Cinema Booking System, the average cohesion of the manual design is higher. The manual design comprises five classes, thus the number of attributes per

class is high. Nevertheless, the manual design also reveals a higher degree of coupling, with thirty six external ‘uses’. For instance, the class “Rule” appears to be tightly

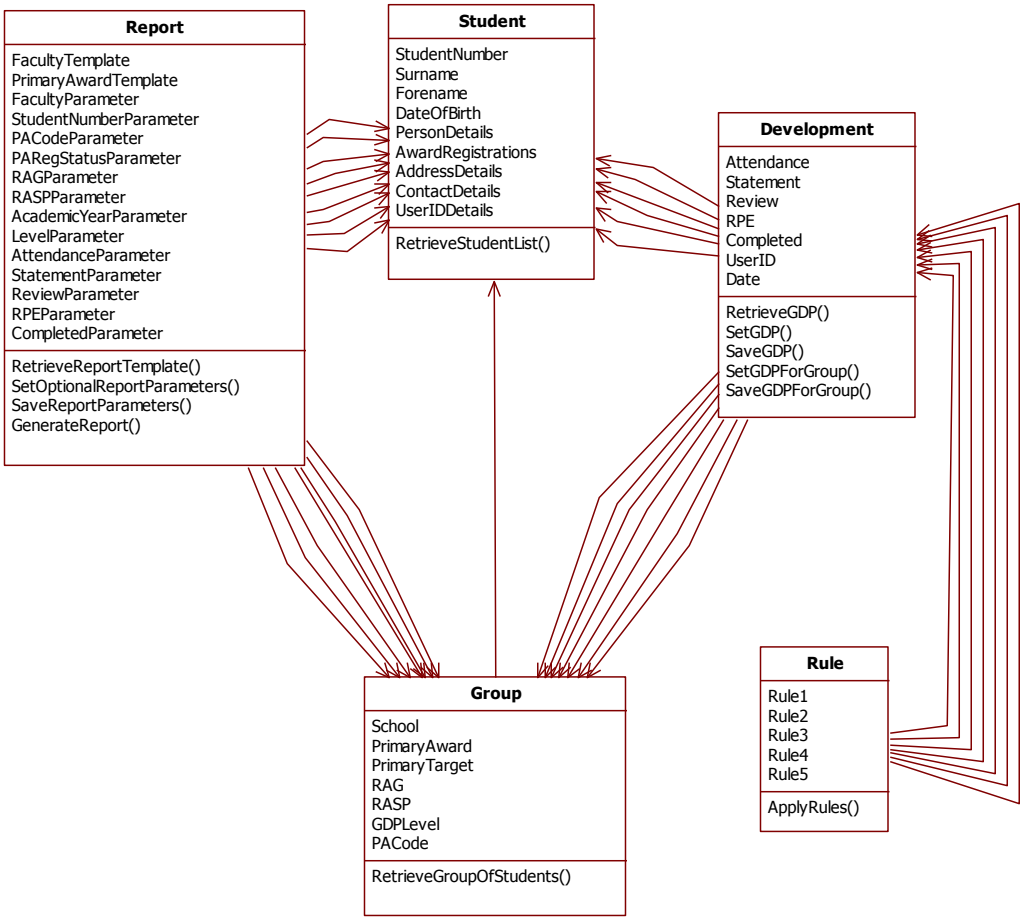

Figure 4.2. Graduate Development Program Manual Design

Table 4.2. Cohesion Values of Manual Design for GDP

Class	Cohesion value
Student	1.0
Development	0.885
Rule	1.0
Group	1.0
Report	0.710
Average	0.919

coupled to the "Development" class, while the "Report" class 'uses' all the attributes of the "Student" and "Group" classes. It seems likely that increased coupling may be an inherent property of a design problem where a larger number of attributes are present. The total number of uses in this design problem specification is 121, of which 36 are 'external' uses. Thus the coupling value of the manual design is 36 / 121, or 0.297. Application of equation (1) from section 3.1 yields a cardinality of the resultant design solution search space derived from this example problem domain of 1.31668×10^{17}.

4.4.3 Select Cruises

The final example design problem is taken from Apperly *et al.* (2003) and is based on an industrial case study relating to a cruise company selling nautical adventure holidays on tall ships in the Pacific Ocean where passengers are members of the crew. Cruises comprise 'legs' or 'passages' from island to island e.g. Auckland to Tonga, Tonga to Vava'u etc. According to Apperly *et al., "Each tall ship has a maximum of twenty-four berths available for guests in addition to twelve for the professional crew. Different types of berth, such as bunk, hammock, cabin or dormitory can be requested by guests".* The resulting computerised system handles quotation requests, cruise reservations, payment and confirmation via paper letter mailing. A detailed specification of the use cases derived from the Select Cruises design problem is available in appendix I. The Select Cruises design problem comprises 30 actions, 52 data and 126 uses.

Adhering to the use case specification, manual class design taken from Apperly *et al.* (2003) is shown in UML notation in figure 4.3. Using the cohesion metric described in the previously, cohesion values of the manual design classes for Select Cruises are shown in table 4.3. The total number of uses in this design problem specification is 126, of which the number of external uses is 57. Thus the coupling value of the manual design is 57 / 126, or 0.452. The manual UML class design comprises 16 classes, a significant increase in design scale compared to the Cinema Booking System or the Graduate Development Program designs. In line with increased scale, an increase in coupling between classes is also evident. Furthermore, it is interesting to note that the manual design "Transaction" class comprises one attribute but no methods, which violates the search representation. This is because in the manual design of Apperly *et al.* (2003), the relationship from the "Enquiry", "Quote" and "Sale" class to the "Transaction" class is one of inheritance. In other words, "Enquiry", "Quote" and "Sale" classes are sub-classes to the "Transaction" super class. Although it is a

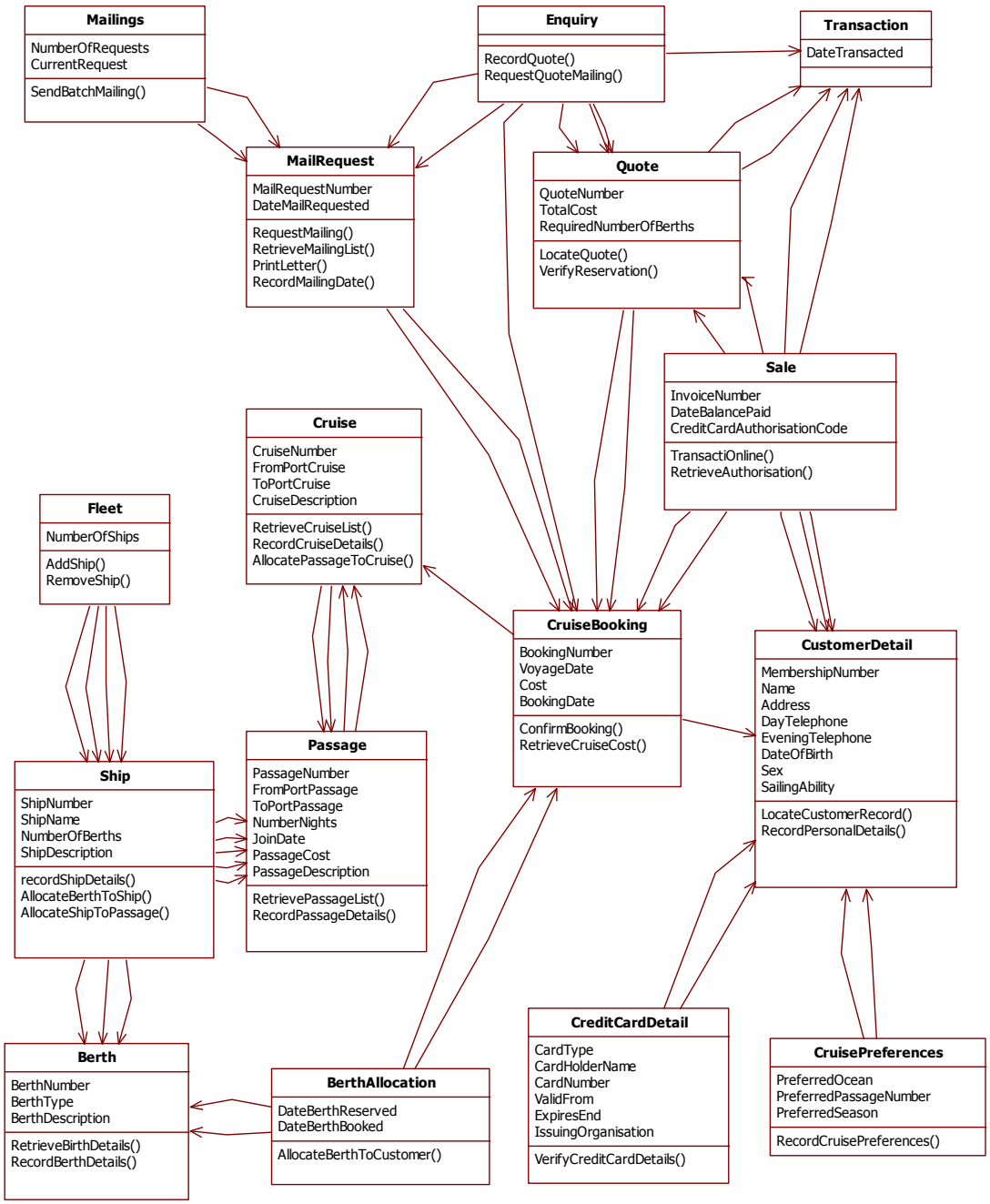

Figure 4.3. Select Cruises Manual Design

limitation of the representation that inheritance relationships are not catered for, it is generally understood that inheritance relationships, while a legitimate feature of the object-oriented paradigm, add to design complexity and also are not as frequently occurring in early lifecycle software designs as ‘use’ coupling. Indeed, in the three software design problems used in this thesis, this in the only instance of the inheritance relationship encountered. Thus as discussed in section 3.1 of the previous chapter of this thesis, this reflects the balance between an appropriate abstract design representation

Table 4.3. Cohesion Values of Manual Design for SC

Class	Cohesion value
Mailings	1.0
MailRequest	1.0
Enquiry	0.0
Quote	0.11
Sale	1.0
Transaction	0
Fleet	1.0
Ship	0.375
Berth	1.0
Cruise	1.0
Passage	1.0
BerthAllocation	1.0
CruiseBooking	0.625
CustomerDetail	0.625
CreditCardDetail	1.0
CruisePreferences	1.0
Average	0.733

and retaining simplicity for the sake of efficient evolutionary search and exploration algorithms. With respect to the cardinality of the search space, the scale appears to be beyond precise computation using a desktop workstation. As equation (1) in the previous chapter states, the cardinality of the search space is defined by:

$$|S| = c!\ \mathrm{S}(a,c)\ S(m,c)$$

Thus an estimate of the order of magnitude of the cardinality has been attempted as follows. Firstly, the number of classes is 16, thus 16! is 2.09227×10^{13}. Secondly, the Stirling number of the second kind for 30 methods and 16 classes has been computed as 2.94081×10^{21}. However, computation of the Stirling number of the second kind for 52 attributes and 16 classes has proved intractable. Therefore, assuming that the Stirling number of the second kind for 52 attributes and 16 classes can be no less than that for the methods, according to equation (1), the cardinality of the search space can be no less than

$$2.09227 \times 10^{13} \times 2.94081 \times 10^{21} \times 2.94081 \times 10^{21}$$

which is

$$1.80947 \times 10^{54}.$$