Designing A New Programming Environment for Randomized Controlled Trials

Jiawang Xing

A thesis submitted in fulfilment of the requirements for the degree of
Master in Software Engineering,
The University of Auckland, 2023.

Abstract

This thesis covers the creation of ISPE-RCT, a programming environment expressly devised for randomized controlled trials (RCTs). This study engages in the innovative intersection of clinical trials, programming languages, and synchronous modelling.

ISPE-RCT's design is underscored by three pivotal features: English-like syntax, built-in SCCharts models, and web services. This triad offers an interactive platform, enabling real-time communication and bridging human languages with programming languages. The SCCharts models further facilitate synchronous behaviors and an effective alert system. The study meticulously investigates the development of clinical trials and programming languages, adding the features of group randomization, data collection, and bias reduction.

Furthermore, ISPE-RCT's fusion of human language and programming underpins its distinctiveness. By harmoniously blending the intuitive expressiveness of human language with the rigorous precision of programming, ISPE-RCT ensures clarity, accessibility, and robustness in RCT designs. Additionally, its synchronous programming approach harnesses the advantages of real-time computation, providing researchers with immediate feedback and dynamic adaptations that traditional programming paradigms might not offer. This synthesis is instrumental in paving a new path for the design and execution of clinical trials, making them more streamlined and precise.

Overall, this study contributes to the confluence of clinical trials, programming, and synchronous modelling. By introducing ISPE-RCT and demonstrating its potential, the research provides a robust foundation for future exploration and offers promising avenues for medical practitioners, researchers, and technology developers. The methods employed in this study also promise to aid in ongoing research, offering a resource to the burgeoning field of medical technology.

Table of Contents

1 Introduction

This chapter will introduce information on clinical trials, current software, and the possible reasons for failures in clinical trials.

1.1 Clinical Trials

1.1.1 Overview of Clinical Trials

Importance of Clinical Trials

Debuted back in the 18th century [1], [2], clinical trials aim to answer specific questions about biomedical or behavioural interventions, thus often incorporating research studies on new treatments and intervention techniques that require additional investigation and comparison. More specifically, a clinical trial includes one or more of following innovations [3], [4]:

- New vaccines;
- New drugs;
- New medical devices;
- New behaviours or lifestyle to promote health;
- New applications of current treatments;
- New approaches to performing surgical procedures.

Researchers usually conduct clinical trials to explore whether certain new treatments and intervention techniques are safe and effective in humans and whether they have fewer adverse side effects than current ones. However, the laws and regulatory bodies governing and regulating clinical trials vary worldwide. New Zealand has its Medicines and Medical Devices Safety Authority (MedSafe), while the United States has its Food and Drug Administration (FDA). Every new treatment or intervention technique has to be approved by a regulatory agency before its wide distribution [5], [6], and a well-designed clinical trial plus other relevant evidence is the most convincing for approval.

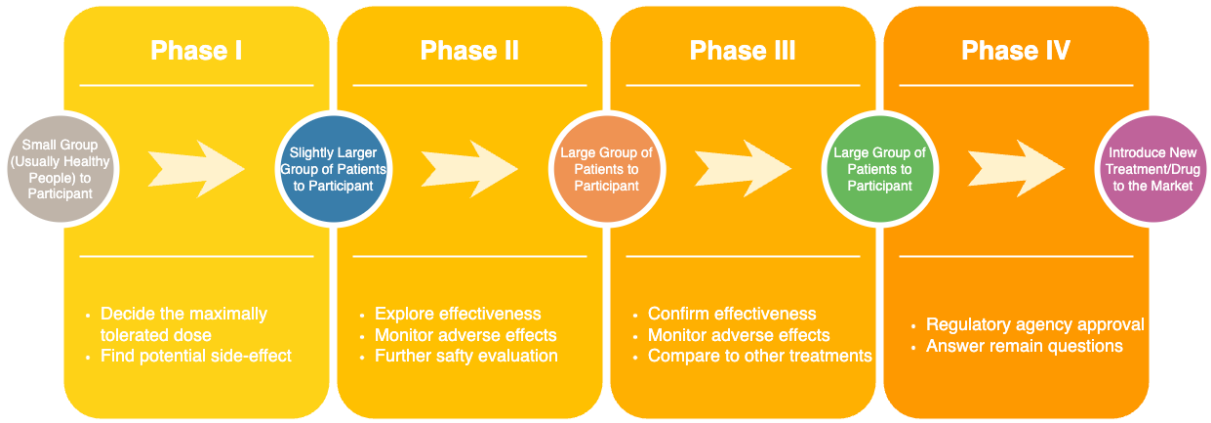


Figure 1.1: Clinical Trial Phases

Clinical Trial Phases

A clinical trial usually has four phases, as shown in Figure 1.1. Although these four phases are not always present in every clinical trial (especially when involving surgeries, new behaviours, or lifestyles), they are the most common ones and, therefore, the typical processes to showcase clinical trials. All data on the typical number of participants for each phase mentioned below came from the NIH website [3].

- **Phase I** often involve a small group (20 to 80) of typically healthy individuals or patients benefiting less from current standard therapies to assess the safety of the drug on trial and estimate its maximum tolerated dose (MTD). Generally, the MTD is determined by administering a small dose at the beginning and increasing the dosage until a pre-specified level of toxicity is observed. The development of clinical trials has yielded various approaches currently adopted to identify MTD. For example, scholars in cancer research often employ the *simple dose-escalation* [7] method based on the criterion that the MTD is reached if around one-third of the participants suffer unacceptable toxicity. As shown in Figure 1.2, researchers taking the dose-escalation approach start by administering a very low dose to a small group of participants (usually three) to keep them safe. If no specified toxicity level is measured, the dose is increased to the next level. Conversely, more participants

(usually another three) are administered with the current dose. If no further toxicity is observed in the additional participants, the dose is increased to the next higher level. If unacceptable toxicity is observed in the additional participants, the current or previous dose is the MTD. Then, Phase II can generally begin.

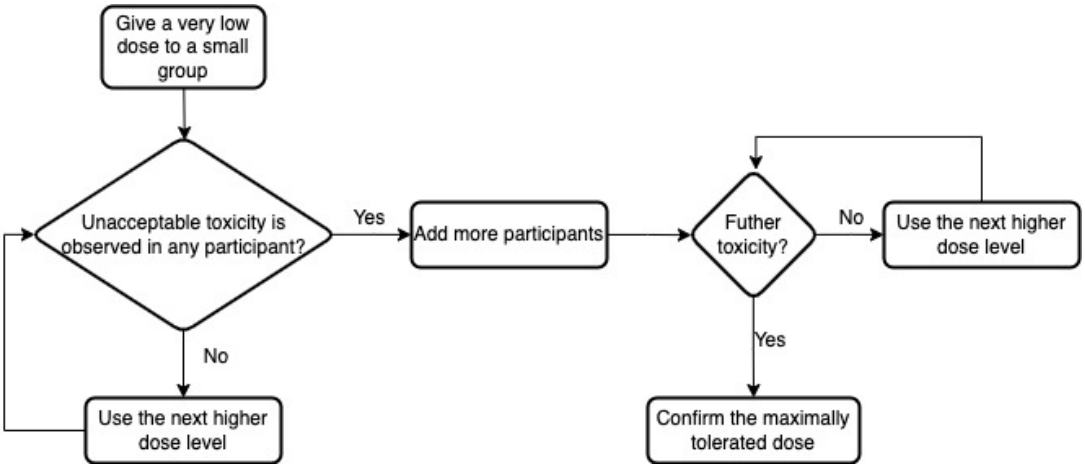


Figure 1.2: The simple dose-escalation Method

- **Phase II** involves a slightly larger group (100 to 300) for additional drug safety testing and medicine effectiveness testing. Researchers usually set concurrent control groups or compare participant statuses before and after treatment to test the effectiveness of the new treatment. By analyzing the comparison results, researchers can assess the effectiveness of the new treatment and determine whether to move forward to the next phase.

- **Phase III** is conducted on a large group (1,000 to 3,000) to learn how different populations react to the new drug. At this phase, researchers often perform Randomized Controlled Trials (RCTs) to collect information and prove the safety and effectiveness of new treatments. More details of RCTs will be discussed in Section 1.2.

- **Phase IV** tests the new drug's effectiveness or safety profile if questions remain unanswered after Medsafe approval [8].

1.1.2 Factors of Failures in Clinical Trials

Despite their decades of clinical trial development, the failure rate has risen sharply in all phases [9]. Only about 10% of clinical trials yielded favourable evidence for the approval of a new drug application (NDA) or biologic license application (BLA) from the regulatory agencies [9], [10]. Approximately 64.5% of the clinical trials have been successful in Phase I, with success rates of 32.4% and 60.1% in Phases II and III, respectively [9], [11].

Schimmel [12] reported the issue of increasing adverse reactions in 1984, yet medical errors still occur daily in hospitals and clinical trials [13], [14]. Landrigan et al. [15] found that the medical specialists' efforts in reducing medical errors improved less until 2010. To make things worse, Hillestad et al. [16] reported that around 440,000 patients died from medical errors in 2013.

Prescription Errors

Many researchers [17]–[20] indicated that the most frequent medical errors originate from prescriptions. Errors in prescribing medicines accounted for 39.1% of all medical errors [21], causing injuries in approximately 1% of the patients. The reasons for prescription errors include illegible words, the research team's writing errors, nurses' understaffing, and overload. For example, physicians writing all prescriptions by hand constitutes a huge workload with lots of repetitive and boring work, which may cause prescription errors.

Absent Monitoring

Monitoring commensurate with treatments is an important factor in a successful clinical trial [22]. However, the monitoring may vary for different participants and scenarios.

Alarm Fatigue

Since clinical trials require real-time participant status monitoring, it is essential to provide medical staff with alerts on adverse and undesired events. Yet, most alarms only emit beeping signals and offer no interaction with the medical staff. On average, the staff may need to handle about 700 monitor alerts per participant per day [23]. Thus, a well-designed clinical trial needs an ordered, timely, and interactable alarm system.

Delayed Reactions

As mentioned above, the reactions of a nurse may be delayed when dealing with hundreds of alarms. Moreover, an unclear prescription may also delay nurse reactions. For example, the prescription often shows only which drug to use but not the specific dosage, making it difficult to react even after selecting the correct drug.

These factors are attributed to the high workload of the medical staff and research team and the existing solutions to these issues will be discussed in Chapter 2. Moreover, other factors may also cause failures in clinical trials.

Safety Issues

Safety is an important issue in the early phases of clinical trials [9]. Researchers have created many technologies to predict the effectiveness and safety of drugs [24]–[27]. Nev-

ertheless, the resultant predictions were not always accurate after the initiation of clinical trials.

Lack of Evidence

The lack of evidence for the effectiveness of new treatments is mainly due to improper designs and unconvincing data analyses, which will be discussed further in Section 1.2.

Summary

In addition, bringing a new drug to market has high expenses. Harrer et al. [28] indicated that on average, a team usually spends 10 to 15 years and 1.5 to 2.0 billion USD on a new drug, with half for the clinical trial phases of the drug development cycle and the other half for the pre-clinical and regulatory processes. Thus, like the implementation of Computerized Physician Order Entry (CPOE), applying Artificial Intelligence (AI) to clinical trials is a great way to reduce medical errors and increase the success rates of clinical trials.

1.1.3 Software used in Clinical Trials

Overview

In past decades, researchers have used many different techniques and software to reduce the workload of the research team and prevent medical errors. Two factors may lead to an unsuccessful clinical trial: the flawed patient recruiting mechanism to find suitable participants and the unmet need for more reliable technical infrastructure supporting treatment management, continuous patient monitor, and enhanced clinical endpoint detection systems. Natural language processing and human-machine interfaces are two techniques that contribute to failure reduction.

CPOE

Although papers on preventing medication errors or prescribing errors in clinical trials are rare, they should also be considered. Since the research team usually monitors 1,000 to 3,000 participants in Phase III, the researchers may experience the same problems as above, rendering it challenging for them to take perfect care of every participant. CPOE is a modern way to prevent medication errors [29] in healthcare and clinical trials. The core component of CPOE is the Clinical Decision Support (CDS), a knowledge-based tool assisting medical staff to determine the treatment and dosage of drugs [30]–[32]. CPOE is designed to help nurses provide the correct treatment to patients based on the knowledge of CDS. However, the evidence that CDS can help to increase medication safety

is lacking [33]. Furthermore, implementing CPOE at a 500-bed hospital costs around 8 million USD plus 1.35 million USD for annual maintenance.

Programming Language

One great solution for the huge amount of repetitive work in clinical trials is to design a proper programming language. The research team can harness the strength of algorithms to address the repetitive work and design appropriate programs to produce prescriptions, monitor participants, send alerts, and interact with nurses. Nowadays, many programming languages are used in clinical trials, such as Python, SAS, R, POP-PL [34], iALARM [35], and SemanticCT [36], each with distinctive advantages. These will be discussed in detail in Chapter 2.

1.2 RCTs

1.2.1 Features of RCTs

RCTs have been the primary clinical trial method for the past few decades [37]. Moreover, in many cases, RCTs are the gold standard to determine the effectiveness of a new treatment. An RCT has the following two key features.

- Being controlled, an RCT usually has the *intervention group* and the *control group*. Researchers simultaneously administer their new treatment to the intervention group and a placebo or the most widely used treatment to the control group, thus rendering the trial concurrently controlled.
- Being randomized, the participants are randomly assigned to either the intervention or control groups and remain unaware of their grouping and the treatment they receive, thus minimizing the bias. In addition, the method of randomization varies from trial to trial, and the easiest and most common method is a "coin toss."

1.2.2 Importance of Statistics

RCTs apply the scientific method in comparing treatments to obtain convincing experimental results, usually to determine if a new treatment outperforms the current one. Researchers generally observe the new treatments' performance in RCTs, which may vary from person to person. Determining whether these different performances are important to the clinical trial is a statistical question.

Fundamentally, clinical trials usually apply the treatment to thousands of participants, and researchers design RCTs to gather empirical evidence from the participants [37]. Using

the empirical evidence, the research team can finally prove or disprove the effectiveness of the new treatment. This explains why statistics and statisticians are important in RCTs.

1.2.3 Bias and Solutions

Overview

RCTs also aim to find the relative advantages of new treatments. If different results are observed in different groups, the team would want to determine whether this difference stems from the different treatments. An overarching goal for the researchers is to create comparable groups regardless of the method used. Like any other statistical issue, sampling errors are inevitable when designing RCTs, which are acceptable [38]. However, other unacceptable systematic effects known as biases should be avoided in the design. Biases in RCTs include selection bias, allocation bias, assessment bias, publication bias, and stopping rules. This study focuses on the first three and their solutions.

Selection Bias and Randomization

Selection bias generally occurs when the participants or the research team know which treatment to receive in advance. Specifically, if the details of the participants are known before their recruitment, the research team may consider the suitability of the participants and select only those contributing to the trial's success.

Many RCTs employ randomization to avoid selection bias [39]. All trial participants are randomly assigned to the intervention group or the control group through randomisation. This is necessary for RCTs due to unconscious or subconscious influences. One way to implement randomization [40] is to record the participants in the patient log upon confirmation of their enrollment in the study and randomize them while ensuring that their records are undeletable.

Allocation Bias and Stratification

Apart from selection bias, many participant factors may also affect their responses to the same therapy [41]–[43], such as sex [44], race [45], self-management [46], and personal status [47]. These factors are likely to cause allocation bias. For example, Hsich et al. [48] indicated that women and men have different survival rates. Therefore, if far more women were assigned to one group than the other, the two groups would not be comparable.

During RCT designing, a standard solution for allocation bias is stratification [49]. Instead of allocating all participants in one step, investigators can assign them depending on their types, known as the prognostic factor. For example, the study by Hsich et al. assigned the participants two different allocations, one for the women and the other for the men.

Assessment Bias, Blindness, and Placebos

During an RCT, investigators may look at objective factors such as deep sleep duration [50] and hemoglobin concentration. Nevertheless, other less objective factors may also be examined, such as the patient's pain level and quality of life [51]. The participants usually provide data on these non-objective factors based on their own subjective feelings. Therefore, if the participants know which treatment they receive in advance, their subjective judgments may be affected, i.e., the assessment bias.

Blinding is an effective method to avoid assessment bias. Suppose the participants know which treatment they will receive before the trial, they may have the psychological implication that the treatment will be effective or ineffective [37]. Thus, many researchers adopt the single-blind or double-blind method when designing trials. In a single-blind trial, participants have no idea which treatment they will receive. A double-blind trial means that neither researchers nor participants know what treatment is being administered.

Placebos are generally drugs similar to authentic treatments but without active ingredients. Placebos effectively avoid the placebo effect [52], [53] and facilitate blindness in RCTs.

Summary

Researchers aim to ensure that the clinical trial can finally get two or more comparable groups regarding these biases and their corresponding solutions. A simple explanation of the concepts in RCTs is as follows.

$$E(X_1) = \mu + \tau_1 \tag{1.1}$$

$$E(X_2) = \mu + \tau_2 \tag{1.2}$$

$$\tau = \tau_1 - \tau_2 \tag{1.3}$$

Assuming that a clinical trial has an intervention group and a control group, with X_1 being the outcome of the intervention group and X_2 being the outcome of the control group. Assuming that the effect of the treatment on different participants is additive. μ is an expected number suggested by researchers during the randomization. Then, $E(X_1)$ represents the expectation of the intervention group's result, and $E(X_2)$ is the expectation of the control group. Equation 1.3 expresses how statisticians calculate the difference between these two groups (also known as the treatment effect [54]).

In other words, statisticians can estimate the treatment effect in a well-designed RCT by calculating $E(X_1) - E(X_2)$. Since μ represents a pre-defined value, the bias from

the above may change the value of μ. In this case, these two groups will not have any comparison, and statisticians will get a biased estimate of τ. Hence, a well-designed RCT must avoid any bias.

1.2.4 Ethical Issues of RCTs

In addition to the design aspects of RCTs, their ethical aspects must also be considered [55]. First, the researchers randomly assign a participant to an intervention or control group to receive an actual treatment or a placebo, which resembles tossing a coin. In this way, the kind of treatment to receive is not a choice. Secondly, since the treatment is still experimental, participants may receive ineffective or poor treatment (even if they are in the intervention group). Before any trials, each participant must be informed of the RCT rules and potential risks or side effects. A patient can only participate with full knowledge of the entire trial background and consent to participate. Additionally, the participants should have the right to withdraw from the trial whenever desired.

1.3 Motivation and Overview

The ultimate goal of a clinical trial is to bring new treatments or interventions to the market for the benefit of as many as possible [8]. The first two sections of this chapter suggest that a clinical trial or RCT is necessary to demonstrate the effectiveness and safety of new treatments and gain regulatory approval. Nevertheless, there is still much to improve in clinical trials.

Efficiency and accuracy are important in clinical trial designs [56]–[58]. The entire design and each clinical trial phase require significant energy, funding, and investments in staffing. Therefore, researchers desire more achievements in less time and greater accuracy, which can be accommodated by computers' efficiency and low error rate. ML and DL provide fine data analysis and data mining tools for statisticians in clinical teams. Knowledge-based tools such as CPOE can help clinical teams to provide treatments based on experience and knowledge. However, many other issues persist in clinical trials. An appropriate programming language is, therefore, a good solution. With the rapid development of computer programs, statisticians can analyze data with the help of programming languages such as Python, R, and SAS. Software developers can design web pages and software for various purposes in languages such as Java, C#, JavaScript, and HTML/CSS. Nonetheless, clinical trial researchers often lack the capacity to program with a suitable language.

The motivation for this study was to develop a new programming environment for most RCTs to reduce the occurrence of potential medical errors and improve the efficiency of clinical trials. This programming environment will not require clinical trial

teams to change their habits but will help them implement and complete their methods more efficiently. It is a natural language-based (similar to English) programming language, and its syntax and semantics are inherited from the prescription writing habits of the researchers. Secondly, the researchers can also write some appropriate code to simultaneously monitor the status of all participants and raise alarms if necessary. In addition, all participants and staff involved in the clinical trials can communicate and interact with each other through the program. A web service is also developed for this purpose. Once the manager has executed the program, the web service can read each patient's treatment and send reminders to the nurses at the appropriate time. The nurses can inform the running program of their task completion through the web service. The web service can also alert the whole research team to adverse events in the patients.

Furthermore, this study will consider the RCT design requirements and ethical issues.

1.4 Scope of This Thesis

This study aims to provide researchers with an easy way to arrange clinical trials or RCTs. By discussing the drawbacks of clinical trials, this study will introduce a solution to the clinical team, an interactive, synchronous programming environment for randomized controlled trials (ISPE-RCT). As a domain-specific programming environment, ISPE-RCT has two main features: interaction and synchronization. Through interaction, the cooperation among the staff and participants involved in clinical trials is improved, while synchronization assists in safely managing the participants. ISPE-RCT aims to offer a structured, predictable, and transparent method for designing and conducting RCTs to reduce errors, enhance efficiency, and ultimately contribute to more reliable and robust clinical trial outcomes. This thesis introduces the implementation of ISPE-RCT and a CPU/memory usage test to simplify the scalability problem. Moreover, a practical evaluation plan is listed in Chapter 9.

- Chapter 1 introduces the relevant research background, including clinical trials and RCTs. Specifically, the clinical trial processes, issues with clinical trials, software currently used in clinical research, and the medical errors in clinical trials are demonstrated. Moreover, RCTs' features, work principles, biases, and ethical issues are discussed.
  By summarising the background and issues related to clinical trials and RCTs, Section 1.4 presents the motivation for this study and a way to overcome the problems.

- Chapter 2 integrates programming languages used in various clinical trials, encompassing data analysis, prescribing, patient condition monitoring, and the provision of alerts. Section 2.4 lists the problem statements by summarising the advantages and limitations of each programming language.

- Chapter 3 presents the syntax and semantics of ISPE-RCT, explicating their resemblance to those in English. The ISPE-RCT language combines the strengths of natural language and programming language. This new language is easy to learn, reusable, time-accurate, and synchronous. This chapter explains in detail how to implement a program by ISPE-RCT and how the syntax and semantics benefit the clinical team. Furthermore, ISPE-RCT is applied to a successful clinical trial to show its precise fitting to RCTs.

- Chapter 4 delves into the technical aspects, including the parser, web service implementation, and core features. The implementation of the parser and web service is demonstrated. Moreover, some core functions, features, and a simulation of a tiny ISPE-RCT program are included.

- In Chapter 5, we delve into comprehensive research tests conducted on several completed clinical trials to ascertain the compatibility and efficiency of ISPE-RCT. This evaluation encompasses two main facets: first, applying the ISPE-RCT language to the trials, serving as a litmus test for its practical functionality, and second, assessing the software's performance by examining its CPU and memory usage. This two-pronged approach ensures a holistic understanding of both the tool's applicability in real-world scenarios and its computational robustness.

- Chapter 7 concludes the achievements in this thesis and and highlights potential avenues for future research.

2

Related Work

This chapter presents the related work on programming languages in clinical trials and an introduction to synchronous programming languages. By summarizing the related work and the advantages of synchronous programming languages, we devised ideas for a new programming environment design and the problem statements.

2.1 Programming Languages in Clinical Trials

2.1.1 Data-oriented Programming Language

As mentioned in Chapter 1, an important approach in clinical trials is comparison. Researchers also use various programming languages in clinical trials, mostly for data analysis. For instance, **Python, SAS, R, Matlab**, etc. A brief introduction to these programming languages is provided here.

Python

Python is a free, multi-purpose, high-level, interpreted programming language first introduced by Guido Van Rossum in 1991 [59]. It has now evolved into a mighty tool for data analysis. The reason for its increasing popularity in data analysis is that it allows analysing data and directly building the corresponding production system. However, the disadvantage of Python is its nature as an interpreted programming language, which makes its code run slower than that of compiled languages like Java and C++. Python now comes in thousands of different packages. **Numpy** (Numerical Python) is

the widely used module of numerical computation, providing data structures, algorithms, and many other functions for numerical data. Apart from that, NumPy can process data stored in Numpy arrays without requiring extra memory to copy data. Many researchers use Numpy for data analysis in clinical trials. **Pandas** is a high-level data analysis tool with useful functions for data input, data manipulation, and data cleaning. Being initially intended for business analytics problems, most functions in Pandas could handle time series data. Moreover, DataFrame in Pandas has similar syntax and semantics to R. **Matplotlib** is a widely used Python module to provide plots and data visualizations [60]. With these three and other powerful modules, Python can easily read and analyze data and generate plots to present the results.

R

Ihaka and Gentleman [61] created the R programming language in 1996 for statistical computing and graphics support at the University of Auckland, New Zealand. Through decades of development, R has evolved into a commonly used, free, open-source programming language in data mining. Bioinformaticians, statisticians, and medical researchers also rely on R to analyze data and develop statistical software. R runs on UNIX platforms, UNIX-similar platforms, Windows, and MacOS, which partially explains its popularity. R also supports data visualization. Moreover, many universities incorporate R in their curriculums. Although R has built-in functionalities, Python performs better in deep learning, benefiting from its packages like OpenNN, Caffe, Keras, etc., and proves more straightforward for designing deep neural networks. Since R can be accessed through the command line interpreter, users have alternative ways to control R objects, including code in Python, Java, C, C++, and .NET.

randomizeR in Clinical Trials

As mentioned in Chapter 1, randomization is essential for clinical trial designs. Uschner et al. [62] designed the randomizeR package in R to support randomization procedures of clinical trials.

$$Y_i \sim N(\mu_A \cdot T_i + \mu_B \cdot (1 - T_i),\ \sigma^2) \tag{2.1}$$

They assume that the sample size of a two-armed clinical trial with parallel group design is N. A and B are treatments that affect the continuous result called Y. A process of randomization M is the possible distribution on the space $\Omega = \{0, 1\}^N$. t_i is a possible value of T_i, and $t_i = 1$ means that the allocation allocates treatment A to participant i, and $t_i = 1$ denotes the allocation of treatment B to participant i, where $t_i \in \Omega$. T_i is the variable from Ω, and y_i (response) results from a normally distributed random variable

with group expectations μ_A, μ_B, and unknown variance $\sigma^2 > 0$.

$$H_0 : \mu_A = \mu_B \quad vs. \quad H_1 : \mu_A \neq \mu_B \tag{2.2}$$

They also tested the null hypothesis, finding no difference between the expectations of A and B.

$$Y_i \sim N(\mu_A \cdot T_i + \mu_B \cdot (1 - T_i) + b_i,\ \sigma^2) \tag{2.3}$$

In Equation 2.3, b_i is the bias of patient i, including selection bias and chronological bias.

The **getAllSeq** function of randomizeR is suitable for the randomization procedures when $N \leq 24$ and the **genSeq** function fits the case when $N > 24$. The **getAllSeq** function can not get a reasonable run-time when $N > 24$. For example, users can generate a complete randomization (CR) for groups A and B in a clinical trial with $N = 20$ using the following code: ' cr' is a constructor function.

```
1  R > N <- 20
2  R > (params <- cr(N))
3  Object of class "cr"
4  design = CR
5  N = 20
6  groups = A B
```

In summary, R is a powerful interpreted language for data analysis with huge built-in libraries.

SAS

Barr [63] first introduced the Statistical Analysis System (SAS) in the 1970s at North Carolina State University. SAS is a practical and comprehensive software for governments, education, finance, and medicine. With solid data analysis capacities, SAS can read data from any file and produce data analysis findings as graphs, tables, and several document formats. Apart from these powerful features, SAS has a simple Graphical User Interface (GUI) for quick access to all such tools. Additionally, coding in SAS resembles writing simple statements, and its syntax is easy to learn. As closed-source software, SAS guarantees data security but is costly on the license [64]. Thus, its applications are limited to large companies, and it is the market leader in enterprise analytics. Some pharmaceutical product development teams have experience with SAS. For example, Li et al. [65] utilized SAS to evaluate Gehan's two-stage design and Simon's two-stage design in sample size computations for phase II clinical trials of anti-tumour medicines. They used the SAS macro tool to compare the advantages and disadvantages of these two approaches. Dmitrienko [66] provided a practical guide for using SAS in clinical trials, demonstrat-

ing the usage of SAS macros, comparisons, and endpoints in clinical trials, data safety, interim data monitoring, and many corresponding examples.

Summary

The most obvious difference among the above languages is that Python and R are open-source while SAS is not. Not considering the budget, SAS is the best choice for research teams. However, most teams choose Python or R for free use of machine learning techniques. Even though many researchers use these data-driven programming languages in clinical trials, most have yet to gain experience in coding. According to the SAS Institute, SAS is easy to learn but has many variables and new syntax. Is there a programming language that is genuinely convenient and easy to learn for these researchers? Yes, Section 2.1.2 shows an English-like programming language focusing on prescription called POP-PL.

2.1.2 High-level Programming Language

High-level programming languages are usually very abstract. In contrast to low-level programming languages, high-level programming languages are commonly user-friendly and more understandable. They aim to provide users with a more straightforward way to design programs or generate programs for specific needs automatically. Thus, high-level programming languages may apply some natural syntax to fit a wide range of users.

Overview

In 2005, Florence et al. [34] created the Patient-Oriented Prescription Programming Language (POP-PL), a high-level programming language designed to express description in medicine with English-like syntax. Florence et al. believe complementary strengths exist between programming languages and natural languages. According to the programming languages and medical research experience, they created POP-PL to help deliver safer and more accurate prescriptions. The idea emerged when the clinical researchers in their team noticed the algorithm for writing prescriptions. Handelsman et al. [67] noticed physicians' and other medical staff's repetitiveness in prescription writing. For example, most prescriptions for women include this line: "Do not use if you are trying to become pregnant during the first six months of pregnancy, except on your doctor's advice." Many physicians use flowcharts, tables, and natural languages to express prescriptions, which unavoidably produce errors and ambiguities. These errors may cause serious harm or even fatality to the patients. As mentioned in Chapter 1, many medical errors and deaths occur during the treatment. Measures to reduce such medical errors include improving the accuracy of the prescriptions, continuously monitoring patient status, establishing clear alerts to adverse events, and reacting faster to the tasks of each patient. POP-PL records all past

events with time series and all issued tasks in a log. In the face of new information, POP-PL reads the log and suggests reactions to relevant medical staff.

Syntax Example of POP-PL

Florence et al. provided an example program written in POP-PL for managing heparin dosage (an anticoagulant).

```
1    initially
2        giveBolus 40 units/kg of HEParin by: iv
3        start 10 units/kg/hour of HEParin by: iv
```

Here is a similar example to demonstrate the POP-PL syntax. A POP-PL script always starts with initialization, including the code to prescribe an initial dose to the patient. The first line encodes the first step of initialization: administering a one-time dose (called Bolus) to the patient with 40 units/kg of heparin. The second line encodes the continuous provision of an intravenous infusion every hour.

```
1  whenever new aPTTResult
2    aPTT < 30              | giveBolus 40 units/kg of: HEParin by: iv
3                           | increase HEParin by: 2 units/kg/hour
4
5    aPTT in 30 to 50       | giveBolus 20 units/kg of: HEParin by: iv
6                           | increase HEParin by: 1 unit/kg/hour
7
8      // aPTT in 50 to 70   Maintain the current HEParin dose
9
10   aPTT in 70 to 90       | decrease HEParin by: 1 unit/kg/hour
11
12   aPTT > 90              | hold HEParin
13                          | after 1 hour
14                          |   restart HEParin
15                          |   decrease HEParin by: 2 units/kg/hour
```

The infusion segment above provides different actions depending on the activated partial thromboplastin time (aPPT) result. The program in this section serves merely as an example, and the range of results and actions may not be suitable for actual use. This segment is similar to a typical if-else statement. When the system gets a new aPPT result, POP-PL determines its range and provides a corresponding action. For example, if the aPPT is below 30, give 40 units/kg Bolus with Heparin and ask the nurse to increase the current dosage of Heparin by 2 units/kg/hour. Like the comments in Java, users can start a comment with "//" in POP-PL. Expressing the range of the result is pretty straightforward, e.g., "aPPT $<$ 30" represents when the aPPT result is below 30, "aPPT

in 30 to 50" expresses that the aPPT result is from 30 to 50, and "aPPT > 90" denotes that the result is above 90. One useful syntax is "**after**," which is used followed by a time constraint and some action expressions to manage the actions afterward.

Summary

POP-PL syntax is obviously English-like, which is indeed easy for anyone to learn, as it is just like writing in English. POP-PL helps physicians monitor participant status, provide suitable prescriptions, and send requests to nurses. Systematic dosage management may reduce medical errors in the treatment. Apart from dosage management, POP-PL also provides an approach to remind nurses to check the aPPT of patients. For example, if a physician sets the program to do the following: check the aPPT result every six hours; either the latest two aPPT values are not in the target range or less than two values are recorded, raise the check message to the doctor. POP-PL also implemented a feature for nurses to interact with the program. After checking the aPPT value of a patient, the nurse can inform the system with a message that the aPPT result is logged some time (1 minute for example) after a treatment. The program will record the result with a timestamp and then apply the treatment to the patient. Thus, the reactions of doctors, programs, nurses, and patients form a positive cycle, leading to fewer medical errors and higher effectiveness.

POP-PL also has other advantages. With the only literal data being numbers, POP-PL provides better algorithms for dealing with different drugs (no need to waste time distinguishing those drugs). Additionally, POP-PL is case-insensitive to facilitate nurses determining which drug to use and avoiding misusing medications with similar spellings. Finally, POP-PL allows physicians various keyword arguments (mainly prepositions) to suit their different writing habits. At the end of their paper, Florence and colleagues tested the participants with different backgrounds to assess whether POP-PL was friendly to those without coding experience. The results showed no relevance between high marks and coding experience, indicating that POP-PL was easy for anyone to learn and use. Therefore, POP-PL is a successful and easy-to-learn high-level programming language. However, as a prescription programming language, it has the following limitations:

Weak interactions among doctors, nurses, and patients.

Interactions in POP-PL can only cover checking results and are limited to sending restricted messages. What if the patient needs to send a message directly to the doctor? What if the patient wishes to send files (e.g., questionnaires) or voice messages to the doctor or nurse? What if the nurse misses the message from the system? There is also no GUI for interactions among all parts of clinical trials. Section 2.1.3 introduces an alert-based Programming Language called iALARM.

Lacking support for Multi-treatments Concurrency.

POP-PL does not allow concurrency. Some scenarios involve synchronizing multiple tasks in clinical trials to reduce medical errors significantly. Suppose the physician needs to administer a specific drug dose to the treatment group and a placebo to the control group. In that case, the program must synchronously handle two groups of participants (send each participant the specified treatment, monitor the status of each participant, and collect data from each participant concurrently). Synchronous programming languages have suitable functions to achieve such features, which will be discussed in Section 2.2.

2.1.3 Monitoring and Alerting Programming Languages

Overview

The causes of medical errors mentioned earlier in this chapter indicate the importance of monitoring patient status and raising alarms in clinical research. It is crucial to alert the clinical team or patients of adverse and life-threatening events. Most of the existing medical monitoring and alert systems are device-oriented [68], [69] and sensor-based [70] whether wearable or wear-less. These systems monitor the signals based on the functionality of the built-in devices and sensors. However, with multiple devices emitting alerts together, the medical staff must spend more time distinguishing and reacting to them, leading to *alerting fatigue*. To avoid *alerting fatigue*, Klimov and Shahar [35] designed the Intelligent Alert Language for Activation, Response, and Monitoring of Medical Alerts (iALARM), a language for the specification of intelligent alerts.

To handle patient data with timestamps, iALARM takes the approach created by Shahar [71], namely, knowledge-based temporal-abstraction (KBTA). Based on the KBTA concept, Boaz and Shahar [72] also presented a temporal-abstraction mediator (Idan), a modular system for addressing abstract, time-oriented medical inquiries in medicine. iALARM is an application of such mediators to compute and abstract the patient's raw measurement data with timestamps so that the monitoring engine can monitor patient status and send iALARM alerts to the medical staff. Moreover, once an alert program is designed, the user can apply it to the whole group.

Schema of iALARM

At the top level, the design of an alert involves Population Expression, Declarative Expression, and Procedural Expression.

The Population Expression tells the system which group of patients to monitor, which can be achieved by directly providing the patient ID or selecting a group of patients according to the context. For example, the team can label each patient and group them.

The Declarative Expression consists of a concept name followed by value constraints and time constraints. Taking the HEParin application in Section 2.1.2 as an example, if the user desires to raise an alert when the aPPT result is above 90 and below 150, the Declarative Expression can be written as follows:

$$< Declarative\ definition > \equiv\ (High\ aPPT\ Result, < 90, 150 >, Now)$$

The user can also replace *Now* with a retrospective constraint, showing the alert's valid time and time duration.

The Procedural Expression specifies the actor to receive the alert, a series of actions or plans to execute when triggered, the significance of sorting the alert's arrays, a clipping expression, and a refractory-period expression.

Lifecycle of an Intelligent Alert

The lifecycle of an intelligent alert, expressed by the clipping expression and refractory-period expression, is quite impressive. When the system captures an abnormal patient state, it raises the first alert, A, which stops after a response time. The lifecycle would terminate if the medical staff responded to it within a time duration. Otherwise, the system waits for one or several spans of the response time (user-defined) and raises the second alert, B. A final alert, C, will be raised if the staff fails to feed an expected response to the system within a response time, and the lifecycle of this intelligent alert ends. The lifecycle design allows users such customized alert management to suit any scenario and avoids long-running alerts or alert fatigue.

Summary

Keeping the KBTA concept in mind, iALARM is a practical and well-designed system to handle the patients' temporal raw data, monitor the patients, and intelligently alert the staff. However, iALARM is only good at monitoring and alerting and is too complex for those without coding experience to learn. Although iALARM can alert an entire group, its intrinsic computation is not synchronous. As a result, iALARM can not execute in a very short time or perform multiple tasks synchronously. In this case, iALARM may not be able to send timely alerts when facing large amounts of actions at the same time. Also, the alert design process is not straightforward for users.

2.2 Synchronous Programming Languages

2.2.1 Embedded Systems, Reactive Systems, and Real-time Systems

Embedded systems are the combination of software and hardware with specific purposes. The functions of an embedded system are usually observing, monitoring, and controlling the connected environment. Such functions distinguish embedded systems from laptops and desktops. Users can reprogram the functions of embedded systems to suit their needs. The hardware part enhances the performance of embedded systems. Moreover, how users program the functions also impacts the performance of the system [73].

Reactive embedded systems constantly react to their environment, which can be the human body or physical processes. Reactive systems continuously receive and compute the inputs from the environment and finally output the result to it. For example, temperature control systems must continuously monitor the temperature, receive the temperature readings regularly, and emit the appropriate signal to help the controller adjust the temperature [74].

Real-time embedded systems focus not only on the functional computations but also on the delay in producing outputs. Like many systems in automatic control, mechanics, and engineering, the scenarios require reactions in a specific time (time constraints) [75].

A hard real-time system has missing absolute deadlines, resulting in system failures. Like many hard real-time systems, real-time embedded systems are usually safety-critical. An error in such systems may cause severe consequences, including money loss, mission failure, and even fatality. Real-time systems are usually classified into asynchronous, pre-estimated time, and synchronous types depending on the different time representations and physical and logical time relationships in their design [76]. Asynchronous models have unknown execution times but deadline constraints. Pre-estimated time models estimate the actual execution time based on previous knowledge. Synchronous models have an abstract execution time, assuming the system always produces a corresponding output of the current input before the following input.

2.2.2 Synchronous Programming Languages

In the early 1980s, synchronous languages emerged to enable the trustworthy design of safety-critical embedded systems. Synchronous languages include SCCharts, Esterel, Lustre, and Signal [77]. When validating and implementing real-time embedded software, synchronous languages are the most design-friendly and proper tool. In the 1980s, researchers found that a synchronous language should have the following features: **concurrency, simplicity, and synchrony** [78].

Concurrency

Synchronous languages must allow functional concurrency with a user-friendly syntax. That is, synchronous languages should have notation block diagrams (dataflow diagrams), hierarchical automata, or some specific syntax corresponding to its field of application. In the early 1990s, users needed an integration of all these different notations. Thus, unified mathematical semantics is essential for synchronous languages.

Simplicity

A synchronous language must have the most straightforward semantics to achieve formal reasoning, as the parallel processes are the fundamental part of a synchronous program.

Synchrony

Synchronous languages must support simple models for the two most commonly used synchronous execution schemes. Event-driven synchronous execution: *for each input event, do execution and update the output.* Sample-driven synchronous execution: *for each clock tick, read input, do execution, and update output.*

Clock and Verification

A synchronous language usually has its own logical time, which divides the time into discrete instants. The system must finish each execution in such an instant to ensure synchrony. Additionally, verification is a commonly required feature of synchronous languages to verify the safety and logical properties. A program is deemed as satisfying the safety property if certain situations never occur. For example, a simple *train-gate* system must avoid making two trains pass through the same junction concurrently. Thus, a system like this must be able to check if two gates are opened simultaneously. These logical properties mean verifying the logical relationship between two events. For example, the *train-gate* system must ensure every train can finally pass the junction.

Summary

A synchronous language system can execute an action in a very short time and, thus, is capable of computing the input instantaneously and reacting to its environment continuously.

2.3 Graphical Languages

2.3.1 Statecharts and SyncCharts

Harel created the Statecharts in 1987 [79], a worldwide popular tool for specifying Real-time and Embedded systems. The visual syntax of Statecharts helped a number of experts from different fields (not only the computer domain) [79]. Moreover, the hierarchy and concurrency of SyncCharts provided a more straightforward and comprehensive way to express complex behaviours. However, Harel did not design a complete syntax set for Statecharts. Even worse, the SyncCharts then could not support determinacy, i.e., the system must always produce the same output sequence when provided an input stimuli sequence. Inspired by the concepts of SyncCharts and the family of synchronous languages, André improved the Statecharts and introduced a statechart-like visual language named Synchronous Charts (SyncCharts) in 1996 [80]. SyncCharts implied a unique value called signal to complete the reaction in a tick, which enabled SyncCharts to achieve determinacy. However, the synchronous model of computation (MoC) was not a perfect approach, as it required one value for each action. For instance, it cannot deal with "if $x > 0$ then $x := 0$" for some shared value x, as the latter would modify a value after it is read. That motivated the development of Sequentially Constructive MoC [81].

2.3.2 Sequentially Constructive Charts

Overview

Von Hanxleden et al. [81] created Sequentially Constructive Charts (SCCharts) in 2014 through a collaboration effort among Kiel University, Bamberg University, and Auckland University. SCCharts is a visual synchronous programming language with sequentially constructive semantics, thus succeeding SyncCharts. SCCharts employ statechart notation and allow deterministic concurrency based on a synchronous MoC without the constraints imposed by earlier synchronous MoCs. SCCharts aims to specify safety-critical reactive systems capable of continuously reacting to internal or external stimuli (the environment). A standard method to achieve this is to divide time into discrete "ticks" and perform a reaction cycle in each tick. The reaction cycles include reading inputs, calling a tick function to achieve a reaction, and writing outputs. The Kiel Integrated Environment for Layout Eclipse RichClient (KIELER) is a research project for enhancing the graphical model-based design of complex systems, and the KIELER compiler provides powerful editing, compiling, and simulation for SCCharts. Users can use KIELER to edit and compile SCCharts code and visualize the model-based approach designed by SCCharts, which reduces the workload for developing real-time reactive systems. Additionally, KIELER also serves control-oriented systems, offering not only concurrency but also shared signals

and non-shared variables. KIELER processes the signals instantaneously to ensure that one thread can communicate back and move on in a tick. This feature gives SCCharts a unique power to complete a reaction in a single tick without breaking up the calculation into multiple ticks.

An ABRO example in SCCharts

An ABRO example of this synchronous language is like the "Hello World" of other modern programming languages.

```
1  scchart ABRO {
2
3    input bool A, B, R
4    output bool O
5
6    initial state ABO{
7      entry do O = false
8
9      initial state waitAB{
10
11       region a:
12       initial state waitA
13       if A go to doneA
14       final state doneA
15
16       region b:
17       initial state waitB
18       if B go to doneB
19       final state doneB
20     }
21
22     do O = true join to done
23     final state done
24   }
25
26  if R abort to ABO
27 }
```

The above is a simple ABRO program written in SCCharts. Every SCChartss model starts with an initial state. In this ABRO program, the initial state is ABO. At the entry, initialize the Boolean "O" as false and move into the "waitAB" state. The logic

of this state is that only when A and B are done can the program move to the next state. As expressed in line 22, it outputs "O" as true and joins to the final state done. Additionally, this program declares a reset input to restart it. Compiling with KIELER yields a corresponding graph shown in Figure 2.1. The chart expresses inputs, outputs, states, regions, and transitions to connect states.

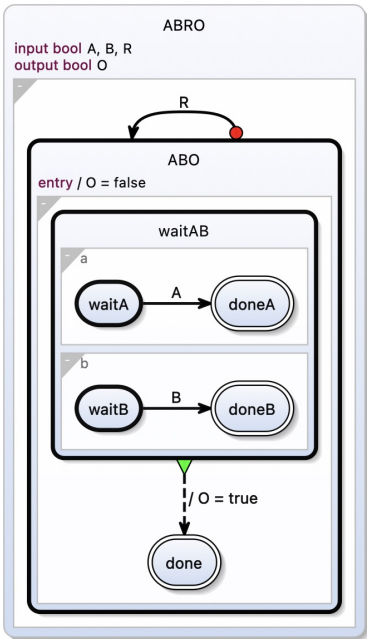


Figure 2.1: ABRO example

States

At the top of Figure 2.1 is the name of the SCCharts program, followed by setting up input booleans and output booleans (like other programming languages, claim before use). In SCCharts, every program must have an **initial state** to tell the system where to start, and users must ensure that each concurrent region has exactly one initial state. Moreover, there are also **final states** marking the end of a thread. The program terminates when reaching the root-level final state. Apart from the initial and final states, other states are simple states with no inner behaviours and can be connected by transitions to other states.

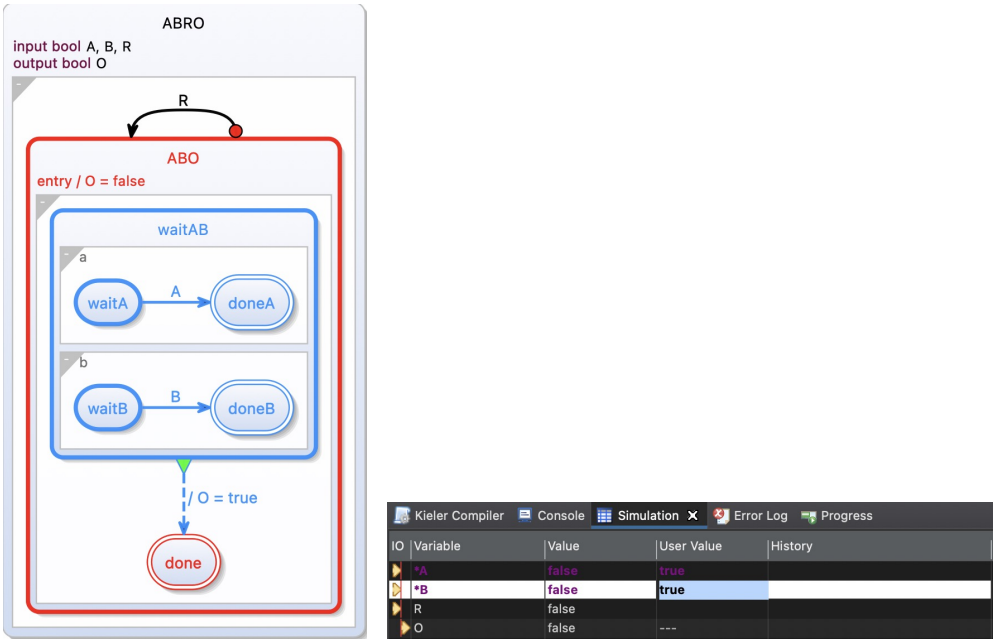
Entry and Abort

As expressed in line 7 of the ABRO program, when moving into a state with **entry**, SCCharts executes the entry action immediately. If there is a prerequisite (optional trigger) before entry, it will only execute the entry action when the trigger is true.

SCCharts has weak aborts and strong aborts. Weak aborts accept the superstate's action in the current tick to move on before termination. However, the strong abort in the example, expressed as a red dot, instantly terminates the current superstate.

Superstates & Regions

The syntax supports nested superstates (using curly brackets after the initial state declaration), each of which creates a new scope. Although any transition preemption can exit the superstate, the most commonly used one is " **join**," as it can "join" the root-level final state once every region reaches its final state. Users can also add action to the join transition, e.g., setting "O = true" in the ABRO example. Regions are the syntax for concurrency models.



(a) Diagram of Simulation (b) Variables & Values

Figure 2.2: Simulation of the ABRO example

Simulation in KIELER

As shown in Figure 2.2, KIELER has a built-in simulation feature for SCCharts to simulate their design process. In the manual option simulation, users can set user values for those input variables and go to the next tick (press the right arrow on the keyboard) to check the result.

2.4 Discussion & Problem Statements

As shown above, researchers indeed need an approach to randomize the grouping, accurately distribute each participant's treatment, promptly administer the doses to the participants, distribute and collect relevant questionnaires in time, and continuously monitor participants' physical conditions. According to the processes of clinical trials and RCTs from Chapter 1, there is a great deal of repetitive work in clinical trials. For example,

the clinical team usually doses the participants and collects questionnaires over the same period, and the participants in the same group typically use the same drug. As a result, doctors must use the same words hundreds of times in the prescriptions. A programming language is a sound approach to reducing workload and error frequency. In addition, medical professionals have long been accustomed to thinking algorithmically in writing prescriptions. Thus, a programming environment specifically adapted to clinical trials is imperative. Data-oriented programming languages (Python, R, SAS, etc.) only benefit physicians with coding experience. POP-PL is a great English-like language but needs to allow more interactions between all parties in clinical trials and provide nurses with more precise, timely, and attention-grabbing alerts. iALARM has a powerful syntax for monitoring patients and raising alerts, which also proves hard to learn. Furthermore, these programming languages need to support synchronous handling of different patients.

Thus, this study aims to design an easy-to-learn, synchrony-supported, and safe high-level programming language with a communication GUI for all users, which can apply to most clinical trials. It will also feature randomized grouping, an English-like syntax compiled into models of SCCharts for different clinical trial scenarios, and a web service for all parties to communicate conveniently. The models support patient status monitoring, specifying dose time and administering method, raising alerts, and especially synchrony.

Additionally, the problem statements are as follows:

- **Research Question 1:** Can we design a new interactive programming environment for clinical trials?
- **Research Question 2:** Can we apply synchronous programming language models to this programming environment to simplify the scalability problem?

3

Syntax and Semantics

This chapter delves into the syntax and semantics of ISPE-RCT. The benefits of this design approach are highlighted, and a case study is provided to illustrate its practical application in a real-life trial.

3.1 Overview

Chapters 1 and 2 emphasize the significance and benefits of high-level syntax in research. Responding to this need, the English-like syntax of ISPE-RCT has been introduced as an innovative solution. Handelsman et al. [67] declared that researchers and physicians do much repeated, complex work when writing prescriptions, which may have a journal article length. Farrell et al. [82]–[84] indicated that the workload of designing a clinical trial is also huge. Therefore, introducing a program like ISPE-RCT has the potential to alleviate these challenges. Doing so enhances researchers' efficiency and allows them to concentrate their expertise on the core objective: crafting the experimental design, unencumbered by monotonous and non-essential tasks.

Previously highlighted was the notable absence of a dedicated programming language tailored for clinical trial design. Compounding this issue, many clinical researchers often lack foundational programming experience. It was against this backdrop that ISPE-RCT was conceived. Its syntax adeptly melds the strengths of both programming and natural languages. This fusion allows researchers to familiarize themselves with its nuances swiftly and employ it in clinical trial design. From an application standpoint, ISPE-RCT streamlines various pivotal processes: it aids research teams in distributing prescriptions,

overseeing participant status, conveying messages during trials, and addressing adverse events. Furthermore, ISPE-RCT is a valuable tool for researchers, ensuring they sidestep biases in RCT designs.

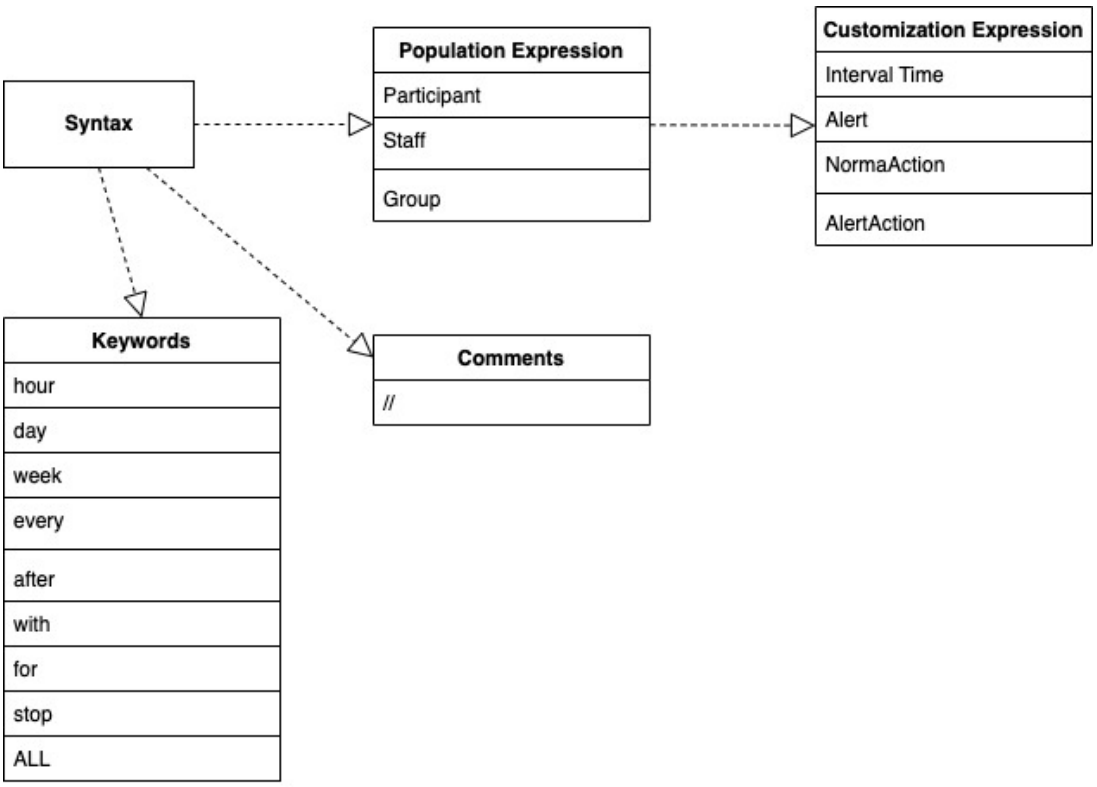


Figure 3.1: Syntax Overview

Figure 3.1 is an overview of the syntax, and the following sections will introduce its syntax and semantics in detail.

3.2 Population Expression

Before officially commencing any clinical trial, the counts of participants, staff, and groups are established. Accordingly, this chapter begins with an exploration of the articulation of populations.

```
1 //Population Expression
2 Participant 20
3 Staff 5
4 Group 2
```

With its English-like design, the syntax ensures that the semantics of population expressions are intuitively understandable. Users can designate quantities using keywords such as 'Participant', 'Staff', and 'Group', succeeded by the corresponding number. For instance, the three selected lines imply that the clinical trial requires twenty participants,

five staff members, and two groups. Crucially, the trial's designer remains uninvolved in participant selection and assignment. This detachment aids in preventing allocation and assessment biases. Furthermore, ISPE-RCT incorporates randomization during the grouping process, serving as a potent countermeasure against selection bias.

3.3 Customization Part

To provide users with a more streamlined and personalized experience, ISPE-RCT incorporates a dedicated customization section. This feature enables users to randomize participant selection, tailor the duration of the clinical trial, and personalize the methodologies employed during the trial. As we transition into the intricacies of clinical trial design, users must specify the trial's duration upfront. This ensures that the trial program operates unobtrusively in the background until its conclusion once activated. This uninterrupted operation is crucial for real-time monitoring of participant status and facilitating immediate interactions when necessary. To further enhance user-friendliness, ISPE-RCT categorizes various trial actions into those necessitating 'Alert' notifications. Users can employ the 'Alert' keyword to define the alert frequency, denoted in seconds.

The keyword 'Alert' in ISPE-RCT syntax is designed to set the alert frequency. For instance, stating 'Alert 5' implies that ISPE-RCT will issue reminders to the staff every 5 seconds until a confirmation of the action's completion is received. The default measurement unit for the 'Alert' keyword is seconds, primarily due to the time-sensitive nature of these AlertActions. The research team aims to ensure such critical actions are promptly executed.

In the syntax of ISPE-RCT, actions necessitating staff intervention are categorized as 'AlertAction,' while the rest are labelled as 'NormalAction.' Drawing from an analysis of prevalent actions in clinical trials as documented in various studies [85]–[93], ISPE-RCT offers predefined NormalActions such as 'logBloodSample,' 'adverseEvent,' and 'collectQuestionnaire' among others. Users can use these actions directly, negating the need for separate declarations. If desired, users also possess the flexibility to define and introduce custom actions into their clinical trials. To implement these, keywords are employed followed by the specified custom actions."

```
//Customization Expression
Interval Time 12 weeks
Alert 5
BasicActions logBloodSample checkESR adverseEvent
AlertActions giveMinocycline givePlacebo collectHAMD
```

3.4 Control Part

The control part stands at the heart of a clinical trial, serving as the locus where ISPE-RCT orchestrates the execution of actions. Drawing inspiration from the synthesis of multiple clinical trials, ISPE-RCT predominantly employs the ensuing syntax to design a clinical trial.

3.4.1 Actions with Nurses

<AlertAction Definition> ≡ (*<Period Expression>* [*AlertAction*] *<Population Expression>* *<Treatment Expression>*),

Where *<Period Expression>* encompasses a set of temporal and value constraints. These constraints delineate when and how frequently the team intends to administer treatments. The [*AlertAction*] is predefined by the user, while the *<Population Expression>* designates the individual or group(s) set to receive the treatments. Furthermore, the *<Treatment Expression>* elucidates the specifics of the treatments in question.

The following subsections introduce these expressions in detail.

Specification of Time Constraint

To support the expression of periodical actions and actions at some specific time, ISPE-RCT mainly has three methods to express the time constraints.

<Period Expression> ≡ (*NULL* | *every <time duration>* | *after <time duration>*), where

- *NULL* stands for acting as soon as the clinical trial starts. In this case, users can skip *<Period Expression>* to [*AlertAction*].
- *every <time duration>* is the constraint to perform a periodical action. For example, if the user uses 'every 2 days' as the time constraint, then ISPE-RCT will run a reminder system for the action every 2 days. Since the action is an [*AlertAction*], the reminder system will remind the staff of every 'Alert interval' to ensure the action is executed promptly.
- *after <time duration>* is the constraint to perform a one-time action after the *<time duration>*. It applies the same *<time duration>* as the previous one, which means it also runs the reminder system, but only for one time.

  <time duration> ≡ [*integer*] [*unit*], where [*unit*] can be measured in minutes, hours, days, and weeks.

Noteworthy, when users use "every" to express *<Period Expression>*, ISPE-RCT initiates the specified action and continues to repeat it at intervals of *every <time duration>*.

Given that section 3.3 has already detailed the declaration of [*AlertAction*] and [*BasicAction*], our discussion will now shift directly to <*Population Expression*>.

Specification of Target Population

To specify the target population, four distinct methods delineate the criteria for selecting participants for whom the action will be executed. Before delving into these methods, it is beneficial to understand the basic concept of participant records. Within ISPE-RCT, each participant is randomly catalogued in the database under code names such as c1, c2, c3, etc. As a result, designers are familiar only with the participant's code name and not their specific identity.

<*Population Expression*> ≡ (*c*<*number*> | *clients from c*<*number*> *to c*<*number*> | *ALL*), where

- *c*<*number*> can represents each participant using different number. When users select fewer participants, they can list them one by one.
- *clients from c*<*number*> *to c*<*number*> returns participants with sequential numbers.
- *group*<*number*> stands for a group of participants. It is worth noting that the number can't be greater than the quantity of 'Group' declared above.
- *ALL* is the symbol containing all the clinical trial participants.

Specification of Treatment Details

The *Treatment Expression* is primarily based on dosage and any changes to that dosage. Other treatments, like dietary therapy, new devices, surgical methods, and lifestyle adjustments, currently do not have specific details that would impact the design of a clinical trial.

<*Treatment Expression*> ≡ (*NULL* | *with* <*number*> [*unit*] <*treatment modification*>), where

- *NULL* is equivalent to an empty string, ''.
- *with* <*number*> [*unit*] represents that users want to give <*number*> [*unit*] drug to the participant(s).
- <*treatment modification*> is useful when the user wants to modify treatments depending on the participant's reflections. <*treatment modification*> ≡ (*NULL* | *If-Else statement*), where *If-Else statement* ≡ (: If/ElseIf [*ActionResult*] <comparison statement>, *dosage change* | Else *dosage change*). The [*ActionResult*] represents

those actions with a numerical return value. So, the <comparison statement> is basically a comparison with a number set by the user, which contains *'greater than,' 'less than,' 'greater than or equal to,' 'less than or equal to,'* and *'equal to'*. Last but not least, *dosage change* consists of 'increase the dosage to *<number>* [*unit*]', 'decrease the dosage to *<number>* [*unit*]', 'keep current doseage'.

A sample program of *AlertAction Expression* is as follows:

```
//AlertAction Expression
every 1 day giveMinocycline for group1 with 100 mg
every 1 day givePlacebo for group2 with 100 mg
collectQuestionnaire for ALL
after 2 weeks collectQuestionnaire for ALL
every 4 weeks collectQuestionnaire for ALL
giveMinocycline to c1 with 100 mg every 1 day:
If collectQuestionnaire > 10, then increase to 150 mg;
ElseIf 7 < collectQuestionnaire < 10, then keep current dosage;
Else decrease to 50mg
```

3.4.2 Actions without Nurses

<BasicAction Definition> ≡ (*<Time Constraint>* [*BasicAction*] *<Population Expression>* | *monitor adverseEvent for <Population Expression>*),

Where *<Time Constraint> and <Population Expression>* are similar to those in the *<AlertAction Definition>*. [*BasicAction*] is pre-defined by the user. Since adverse events have the first priority to avoid, ISPE-RCT has a preset model to monitor any adverse events. Notice that any ISPE-RCT programs should end with the keyword 'stop.'

```
//BasicAction Expression
logBloodSample for ALL
checkESR for ALL
monitor adverseEvent for ALL
stop
```

3.5 A case study - Minocycline as an adjunct for treatment-resistant

3.5.1 Introduction

Trivedi et al. [85] released a clinical trial, Establishing moderators and biosignatures of antidepressant response in clinical care (EMBARC), to identify moderators and mediators

of treatment response for depression. Sertraline (SERT), a serotonergic antidepressant, and bupropion (BUP), a nonserotonergic antidepressant, are two active treatment arms of this trial. Regarding figure 3.2, EMBARC has two stages, each lasting eight weeks. Trivedi et al. used the Hamilton Rating Scale for Depression 17-item [HRSD-17] to measure the primary outcomes. Moreover, the target sample of EMBARC is 300.

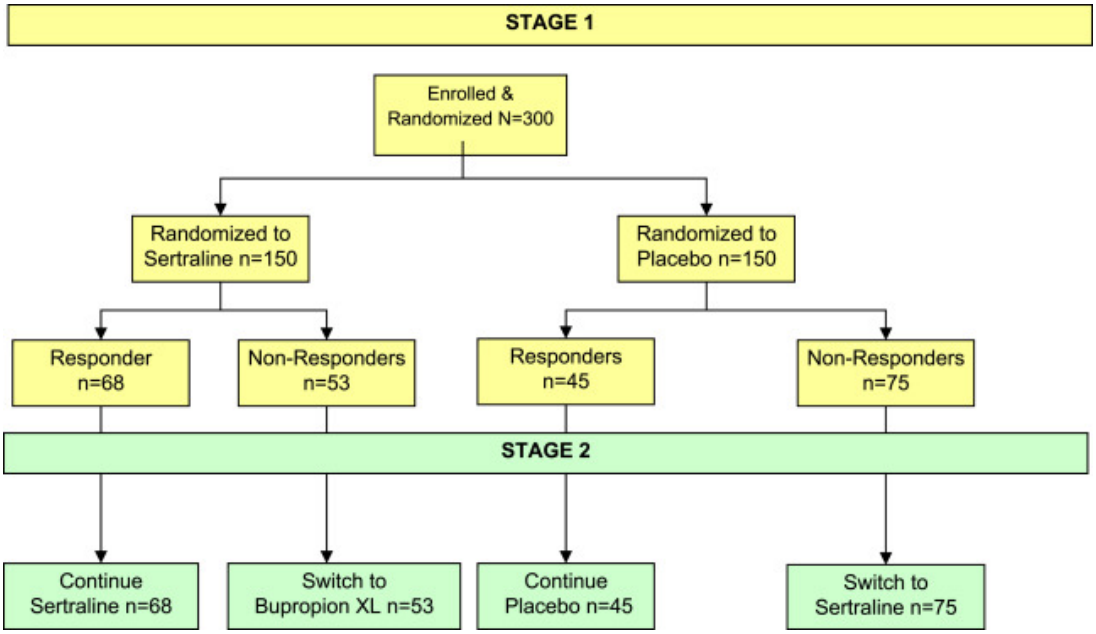


Figure 3.2: Study Design of EMBARC

EMBARC Stage 1 Treatment

Over the initial eight weeks, participants are divided into two groups, with each group comprising 150 participants. These groups receive treatments of SERT and a placebo, respectively. After week 8, participants undergo an assessment using the Clinical Global Improvement (CGI) scale. Those who score below the "much improved" threshold are categorized as non-responders. Subsequently, the research team administers BUP treatment to these non-responders.

EMBARC Stage 2 Treatment

In stage 2, SERT non-responders receive BUP, and placebo non-responders receive SERT. In this stage, the research team can compare the difference in medicinal effects and side effects between SERT and BUP.

3.5.2 User Story

This is the user story of important actions in EMBARC. Some similar actions within the same period are not mentioned here.

EMBARC Stage 1 User Story

- As the leader of EMBARC, I want the first stage to last eight weeks.
- As the leader of EMBARC, I want to allocate 300 participants into randomly two groups so our team can investigate SERT's effects and side effects.
- As the leader of EMBARC, I want to give group 1 SERT 100 mg per day. If a participant's status significantly improves (assessed by [HRSD-17]), our team will reduce the dosage to 50 mg daily. If a participant's status has almost no change, the dosage will be increased to 200 mg per day. Otherwise, the participants should keep taking the current dosage. Thus, each participant can get the most customized dosage.
- As the leader of EMBARC, I want to give group 2 place two pills (50 mg each) per day. If a participant's status significantly improves (assessed by [HRSD-17]), our team will reduce the dosage to 1 pill daily. If a participant's status has almost no change, the dosage will be increased to 4 pills per day. Otherwise, the participants should keep taking the current dosage. Thus, the bias can be minimized.
- As the leader of EMBARC, I want to remind the staff of each treatment every five seconds to ensure treatments can be applied regularly and on time so system differences can be minimized.
- As the leader of EMBARC, I want to monitor the adverse events of every participant so that we can ensure safety and security.
- As the leader of EMBARC, I want to collect the blood sample of each participant on weeks 1, 4, and 8 so we can record data for future analysis.
- As the leader of EMBARC, I want to collect each participant's electroencephalogram (EEG) on week 1 so we can record the data we need.

EMBARC Stage 2 User Story

- As the leader of EMBARC, I want the second stage to last eight weeks.
- As the leader of EMBARC, I want to treat CERT to CERT responders and placebo non-responders. In addition, I want to switch SERT non-responders to BUP XL and switch placebo non-responders to SERT. Thus, our team can compare the different effects between SERT and BUP.
- As the leader of EMBARC, I want to give group 1 SERT 100 mg per day. If a participant's status significantly improves (assessed by [HRSD-17]), our team will

reduce the dosage to 50 mg daily. If a participant's status has almost no change, the dosage will be increased to 200 mg per day. Otherwise, the participants should keep taking the current dosage. Thus, each participant can get the most customized dosage.

- As the leader of EMBARC, I want to give group 1 BUP 300 mg per day. If a participant's status significantly improves (assessed by [HRSD-17]), our team will reduce the dosage to 150 mg daily. If a participant's status has almost no change, the dosage will be increased to 450 mg daily. Otherwise, the participants should keep taking the current dosage. Thus, each participant can get the most customized dosage.
- Other requirements are similar to stage 1.

Above are the user stories based on the designer's perspective. Since the user story of other staff and participants needs the knowledge of ISPE-RCT, Chapter 4 will explain it in detail.

3.5.3 A Program Example

Program of EMBARC Stage 1

```
1  //Program for EMBARC Stage 1
2  //Population Expression
3  Participant 300
4  Staff 4
5  Group 2
6  //Customization Expression
7  Interval Time 8 weeks
8  Alert 5
9  BasicActions logEEG
10 AlertActions giveCERT givePlacebo collectHAMD collectBloodSample
11 //AlertAction Expression
12 every 1 day giveCERT for group1 with 100 mg:
13 If collectHAMD > 15, then increase to 200 mg
14 ElseIf collectHamd <= 8, then decrease to 50 mg
15 Else keep current dosage
16 every 1 day givePlacebo for group2 with 100 mg:
17 If collectHAMD > 15, then increase to 200 mg
18 ElseIf collectHamd <= 8, then decrease to 50 mg
19 Else keep current dosage
```

```
every 1 day collectHAMD for ALL
every 4 weeks collectBloodSample for ALL
//BasicAction Expression
logEEG for ALL
stop
```

Program of EMBARC Stage 2

```
//Program for EMBARC Stage 2
//Population Expression
Participant 300
Staff 4
Group 3
//Customization Expression
Interval Time 8 weeks
Alert 5
BasicActions logEEG
AlertActions giveCERT givePlacebo giveBUP collectHAMD
collectBloodSample
//AlertAction Expression
every 1 day giveCERT for group1 with 100 mg:
If collectHAMD > 15, then increase to 200 mg
ElseIf collectHamd <= 8, then decrease to 50 mg
Else keep current dosage
every 1 day giveBUP for group2 with 300 mg:
If collectHAMD > 15, then increase to 450 mg
ElseIf collectHamd <= 8, then decrease to 150 mg
Else keep current dosage
every 1 day givePlacebo for group1 with 100 mg:
If collectHAMD > 15, then increase to 200 mg
ElseIf collectHamd <= 8, then decrease to 50 mg
Else keep current dosage
every 1 day collectHAMD for ALL
every 4 weeks collectBloodSample for ALL
//BasicAction Expression
logEEG for ALL
stop
```

3.5.4 Summary

This chapter delved into the syntax and semantics of ISPE-RCT, providing an illustrative case study for clarification. Fundamentally, ISPE-RCT is equipped to address all clinical trial requirements. The expressions discussed earlier offer flexibility, allowing users to rearrange the order of priority as per their preferences. Additionally, every preposition serves as a directive for the compiler, granting users the liberty to modify the sequence of prepositions and their associated elements. In doing so, ISPE-RCT offers a highly customizable syntax tailored to individual needs. For instance, the *<Population Expression>* of EMBARC Stage 1 can be rephrased as follows:

```
1 //Alternative Population Expression for EMBARC Stage 2
2 Staff 4
3 Participant 300
4 Group 3
```

Advantages of Syntax

- The clinical trial designed by ISPE-RCT is logical, accurate, personalized, and reusable.
- With ISPE-RCT, designing a clinical trial becomes as intuitive as writing in English. Anyone proficient in English can quickly adapt to ISPE-RCT.
- ISPE-RCT can help avoid selection and allocation biases by double-blind random allocation. (The design to avoid biases will be mentioned in chapter 4.)
- The customization part of ISPE-RCT helps it to adapt to a variety of different clinical trials.

Disadvantages of Syntax

While participants cannot be reassigned to different groups in stage 2 based on their performance in stage 1 of EMBARC, the allocation for stage 2 naturally follows that of stage 1, ensuring no selection bias. Consequently, the research team can alternate allocating participants between the groups. Further discussion on this topic will be presented in chapter 7.

4

Implementation

In this chapter, we delve into the implementation of ISPE-RCT. It has a parser that seamlessly translates the English-like syntax into SCCharts models and a web service facilitating communication between staff and participants. By exploring this chapter, readers understand ISPE-RCT's design, implementation, and operational processes comprehensively. This exploration offers valuable insights into its benefits and guides readers on optimal utilization.

4.1 Overview

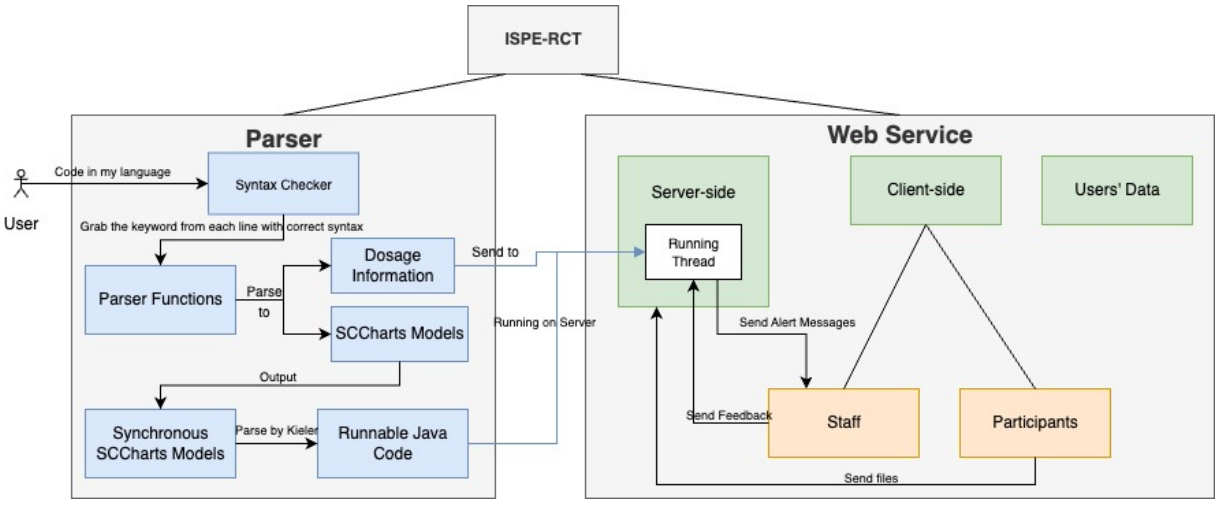


Figure 4.1: Overview of ISPE-RCT Design

Figure 4.1 provides a comprehensive overview of the design. In the parser component, users draft a program using the ISPE-RCT language, which ISPE-RCT subsequently processes. The parser then translates this high-level language into SCCharts models and associated action information. Ultimately, ISPE-RCT generates a final synchronous SCCharts model executed in its Java version.

Concurrently, the Web Service runs the Java model in the 'Running' thread, facilitating communication by transmitting messages to the staff or the system. Staff, acting as clients, can relay an 'ActionDone' message back to the server. Participants, on the other hand, can transmit files to the server. Additionally, ISPE-RCT employs MySQL to log user details and messages.

4.2 Back-end Framework

This section delves into the implementation of the compiler, elucidating its foundational algorithms. It will also shed light on the design of the SCCharts models and offer insights into the simulation of each. Through this discussion, readers will gain a deeper appreciation of the strengths and benefits of ISPE-RCT.

4.3 Parser

The parser is a compiler that recognizes the keywords delineated in Chapter 3. It extracts the data following these keywords and subsequently constructs a SCCharts model corresponding to each action. Importantly, every action possesses its dedicated SCCharts model to oversee its operation.

4.3.1 Algorithms

The algorithms embedded within ISPE-RCT form the foundation for translating human language into a synchronous programming language. The overarching algorithm encompasses multiple functions, each dedicated to solving a specific expression, offering the flexibility of interchangeable declaration orders. Furthermore, an integrated syntax verification system scrutinizes each user-designed syntax line, pinpointing the precise location of any discrepancies. The following subsections will elucidate these aspects in detail.

Population Algorithm

The population algorithm corresponds to its syntax explanation, managing the randomization and the participants' allocation of the clinical trial.

```
1 //Population Algorithm
```

```
2  public List<List<String>> population(int check, String[] input){
3      //Check Initialization
4      if(input[check] == null || input[check+1] == null)
5      {
6          System.out.println("Please initialize staff and
7          particiapnts."); return;
8      }
9      //Check initialization and set up the staff and participants
10     int len1 = intValueOf(input[check].split(" ")[1]);
11     int len2 = intValueOf(input[check+1].split(" ")[1]);
12     //Set up return List
13     List<List<String>> populationList = new
14     ArrayList<List<String>>();
15     if(input[check].toUpperCase().contains("STAFF") &&
16     input[check+1].toUpperCase().contains("PARTICIPANT")) {
17         check = check + 2;
18         //Initial Staff set
19         for(int i = 0; i < len1; i++){
20             staff[i] = "s" + String.valueOf(i+1);
21         }
22         //Initial Participants set
23         for(int i = 0; i < len2; i++){
24             participants[i] = "p" + String.valueOf(i+1);
25         }
26         populationList.add(staff);
27         populationList.add(participants);
28     }
29     else if(input[check].toUpperCase().contains("PARTICIPANT")
30     &&input[check+1].toUpperCase().contains("STAFF")) {
31         ...
32     }
33     else{System.out.println("Please check the syntax at
34     initializing the Participants and Staff."); return;}
35     return populationList;
36
37 //Integer checking
38     public static Integer intValueOf(String s) throws
39     NumberFormatException{
40         return Integer.parseInt(s,10);
```

```
41      }
```

In this section, the variable 'check' represents the current line of the user's program.

As illustrated, each phase incorporates syntax verification to alert users of potential errors and facilitate a seamless program design. The process begins with an assessment to ensure valid content within the user's input (lines 4 - 8). For example, if the user inputs empty lines of the staff and participants' initialization, the parser will print a line to remind the user to initialize staff and participants first.

Lines 10 and 11 extract the counts of staff and participants. Subsequently, lines 13 through 29 are employed to construct the population list. During this process, the staff and participant sets are initialized, with each individual's unique code name being integrated into the respective set. Upon completing this procedure, the sets for both staff and participants are added to the overall population list, which is subsequently returned in line 35. The functionality encapsulated by the 'else if' conditions from lines 29 to 32 mirrors the preceding steps, and for brevity, a detailed description has been excluded. ISPE-RCT has mechanisms to mitigate issues arising from empty inputs, missing keywords, or non-integer inputs. The system also has a feature that checks if the string succeeding the terms 'Staff' or 'Participant' can be translated into an integer; failing this, a NumberFormatException is invoked. The methodology for this integer validation is delineated in lines 37 to 41.

In essence, this 'population function' is adept at ensuring the validity of the provided input lines while concurrently crafting a comprehensive list comprising staff and participants.

```
1  //Random groups setup
2  //Get a Random number
3  public static int getRandom(int i) {
4      Random r = new Random();
5      return r.nextInt(i);
6  }
7  public static List<List<String>> randomization(String[]
8  input, int check, List<List<String>> populationList) {
9      String [] subinput3 = populationList[1];
10     int groups = intValueOf(input[check].split(" ")[1]);
11     // Validation
12     if (subinput3.length < (groups * 2)) {
13         System.out.println("You can only have" +
14         (input.length / 2) + "at most.");
15         return null;
16     }
```

```
17      if (groups < 1) {
18          System.out.println("Group should be at least 1.");
19          return null;
20      }
21      List<String> list = new ArrayList<String>();
22      List<List<String>> groupsList =
23      new ArrayList<List<String>>();
24      for (int i = 0; i < subinput3.length; i++) {
25          list.add(subinput3[i]);
26      }
27      Collections.shuffle(list);
28      int participants = subinput3.length / groups;
29      // Start grouping
30      for (int i = 0; i < groups; i++) {
31          List<String> group = new ArrayList<String>();
32          for (int j = 0; j < participants; j++) {
33              int random = getRandom(list.size());
34              group.add(list.get(random));
35              list.remove(random);
36          }
37          groupsList.add(group);
38      }
39      // Allocate participants left again
40      for (int i = 0; i < list.size(); i++) {
41          groupsList.get(i).add(list.get(i));
42      }
43      return groupsList;
44  }
```

The function 'getRandom', delineated in lines 3 to 6, leverages Java's intrinsic Random functionality, returning a random integer utilized in group allocation.

Lines 9 and 10 instantiate an integer variable, 'groups', to ascertain the number of distinct groups. Line 12 to 20 is a validation of the number of groups. The permissible range for the number of groups, contingent upon the participant count, lies between zero and half the total number of participants. If the user tries to allocate participants into an invalid number of groups, the system will raise an alert message. For example, suppose the user wants a number greater than half of the participants. This validation will tell the user the maximum number of groups based on the previous participants' input. Moreover, if the input number is less than 1 or not a valid integer, it will remind the user to allocate at least one group.

Line 21 constructs a string list — referred to as the 'input list' in subsequent explanations — intended to store participant data derived from the population list. Meanwhile, line 22 spawns a new string list, designated as the 'output list'. The segment from lines 24 to 26 extracts data from the population list, populating the input list. Line 27 employs the 'shuffle' function to randomize the input list's order, while line 28 discerns each group's intended size. The crux of the grouping mechanism, spanning lines 29 to 38, effectively distributes participants across groups. Assuming we are planning to arrange three groups, lines 30 and 31 will create three empty group lists. For each of these group lists, we'll use the random function to get a random number greater than 0 and less than the size of the group list. The function then uses this random number to get a random number from the input list, add it to the group, and remove it from the input list to keep the consistency of the input list size. Finally, once each group reaches its size, the function will add each group to the final group list.

Considering that the participant count might not seamlessly be divisible by the group number, the segment from lines 39 to 43 ensures equitable distribution of any remaining participants. Here, a loop incrementally adds these residual participants across groups.

In summation, ISPE-RCT capably facilitates randomized group allocation within clinical trials through this randomisation function.

Customization Algorithm

The customization algorithm encompasses the initialization of the clinical trial and the determination of the alert time interval. In addition, it aids in the configuration of basic and alert actions. To illustrate how a user defines the alert time interval, it mandates the utilization of a specific keyword accompanied by an integer. This prescribed format ensures clarity and consistency in user declarations, streamlining the setup process for the clinical trial.

```
1  //Initialize the alert time interval
2      public static int alertInit(String [] input, int check){
3          if(!input[check].toUpperCase().contains("ALERT")){
4              System.out.println("You need to Initialize the alert
5              time interval.");
6              return 0;
7          }
8          else{
9              if(intValueOf(input[0].split(" ")[1]) == 0){
10                 System.out.println("The alert time interval
11                 should be at least 1 minute")
12             }
```

```
13              else{
14              return intValueOf(input[0].split(" ")[1]);check++;
15              }
16          }
17      }
```

The approach taken for the alert time interval, as delineated in lines 3 to 7, mirrors the methodology used in the population function. Specifically, the system checks for the keyword 'Alert'. If a user neglects to specify the alert time interval, this function will prompt the user to provide the necessary information. Moreover, should a user input an interval time of less than one minute, the system will recommend adjusting this duration to a minimum of one minute. After successfully passing these validation checks, the system will convert the input into an integer value and subsequently return it.

Notably, the sequences for declaring basic actions and alert actions possess a degree of flexibility, allowing users to interchange their order. This has resulted in some repetition within the code dedicated to action initialization. As such, for the sake of brevity, we have chosen to omit the repetitive code found in line 26. It is recommended that users group their declarations of alerts and basic actions. The procedure for initializing actions is detailed in the following sections.

```
1  //Initial the basicActions and alertActions
2  public static List<List<String>> actionInit(String[] input, int
3      check, List<String> basicActions, List<String> alertActions){
4      List<List<String>> actionsList=new ArrayList<List<String>>();
5      if(input[check].toUpperCase().contains("ALERTACTIONS")
6  && input[check+1].toUpperCase().contains("BASICACTIONS")){
7          String [] subinput4 = input[check].split(" ");
8          String [] subinput5 = input[check+1].split(" ");
9          for(int i = 1; i < subinput4.length; i++){
10                 if(!alertActions.contains(subinput4[i])){
11                     alertActions.add(subinput4[i]);}
12         }
13         for(int i = 1; i < subinput5.length; i++){
14                 if(!basicActions.contains(subinput5[i])){
15                     basicActions.add(subinput5[i]);}
16         }
17         actionsList.add(basicActions);
18         actionsList.add(alertActions);
19         check = check + 2;
20     }
```

```
21      else if(input[check].toUpperCase().contains
22 ("BASICACTIONS") && input[check+1].toUpperCase().contains
23 ("ALERTACTIONS")){
24              ...
25      }
26      else{System.out.println("Please check the syntax at
27      initializing the AlertActions and BasicActions.");}
28      return actionsList;
29 }
```

In the provided algorithm, line 4 initializes the return action list. The subsequent lines, 5 and 6, assess the presence of the keywords 'alertAction' and 'basicAction', which are integral to the process. With these detected, lists for both alert and basic actions are declared. Specifically, lines 9 to 10 and 13 to 14 read the respective alerts and basic actions. Following this, lines 11 and 15 incorporate these actions into their respective lists. To compile these segmented actions, they are combined into a comprehensive action list. It is also crucial to note the update to the 'check' counter in line 19.

The segment from lines 21 to 25 caters to the scenario with an alternate order of basic and alert action inputs. In cases where the input is either invalid or devoid of the essential keywords ('alertAction' and 'basicAction'), the user is alerted to review their syntax, as indicated by lines 26 and 27. The final action list is returned in line 28 after successfully validating all parameters.

Actions Control Algorithm

Given the intricacy of the action statements, the system incorporates multiple functions dedicated to syntax verification. Each function is designed to identify the index of specific keywords, streamlining the checking process. These functions are strategically invoked to facilitate the translation from the high-level language to the SCCharts programming language.

```
1  //Actions Control Function
2  public String getPeriod(int index, String strr, int check){
3      if(strr[index].toLowerCase().equals("every") ||
4          strr[index].toLowerCase().equals("after")){
5          int timer = intValueOf(strr[index+1]);
6          if(strr[index+2].toLowerCase().contains("minute") ||
7          strr[index+2].toLowerCase().contains("hour") ||
8          strr[index+2].toLowerCase().contains("day")){
9              if(strr[index+2].toLowerCase().contains("hour"))
10             {timer = timer * 60;}
```

```
11              else if(strr[index+2].toLowerCase().contains("day"))
12              {timer = timer * 1440;}
13          index += 3;
14          return strr[index] + timer + " " + index;
15          }
16          else{System.out.println(String.format("Error on line %s,
17          Please check your syntax of time units.", check));
18          return;}
19      }
20  }
```

The function designed to check period expressions returns both a keyword and an integer value, with the latter representing the duration in seconds. The variable, denoted as 'strr', corresponds to the current action expression line, while 'index' is a global variable aiding the system in tracking the current position.

The function assesses user input for the presence of keywords 'every' or 'after', as seen in lines 3 and 4. A 'timer' variable is initialized in line 5 to validate and store the numeric value succeeding these keywords. If the user doesn't input a valid integer, an error message is issued, as delineated in lines 16 and 17, leading the function to terminate at line 18. However, if the timer is a valid integer, we will check the unit following the timer. Lines 6 to 8 state the verification of unit keywords. If the unit is 'hour', we will multiply the timer by 60 to convert it from hour to minute. If the unit is 'day', it multiplies 1440 to the timer to convert it from day to minute. We convert all the possible units to minutes because all SCCharts models use one minute as a tick.

```
1  public String getMembers(int index, String strr, int line){
2      if(!strr[index].toLowerCase().equals("for"))
3      {System.out.println(String.format("Error on line %s,
4      Please check your syntax of action.", line)); return;}
5      //get members
6      if(strr[index+1].toUpperCase().contains("GROUP"))
7          {type = 1; return type+' '+strr[index+1];}
8      else if(strr[index+1].toUpperCase().contains("ALL"))
9          {type = 2; return type+' '+strr[index+1];}
10     else if(arraySearch(participants, strr[index+1]) != -1)
11         {type = 3; return type+' '+strr[index+1];}
12     else{System.out.println(String.format("Error on line %s,
13     Please check the name of client.", line)); return;}
14 }
15 public String getDosage(int index, String strr, int line){
```

```
16      if(strr[index].toLowerCase().equals("with")){
17          dosage = Float.parseFloat(strr[index+1);
18      }
19      else{System.out.println(String.format("Error on line %s,
20      Please check your syntax of dosage.", line)); return;}
21  }
```

The functions described above extract data from any given action definitions. The 'getMembers' function manifests in three distinct forms, each representing a unique method to identify participants designated for a particular treatment. Each function returns the corresponding string that ISPE-RCT can use later. Line 3 checks if the line contains 'for' at the correct position. We'll print an alert message and stop getting members if there is an error. Lines 6 to 11 read the participants that will receive the intervention. For example, if the user wants to give intervention to a group, we'll record it as type one and return the type and group name. 'All' is type 2, and clients with code name is type 3. In this case, the main parse function can read the type first and then use a proper function to parse it to a specific participant set.

The function 'getDosage' reads the keyword 'with' at line 16 and parses the dosage to a float number at line 17. However, if the dosage is an invalid number, this function will refuse to read it and raise an alert message by the code at lines 19 and 20. Thus, we can ensure the SCCharts will get the correct variable type and value.

4.3.2 Parser

This section delves into the parser algorithm responsible for converting the previously stored data into an SCCharts program. The prior functions effectively capture and store the data from each line by employing a for loop.

Every Case

```
1  public static void every_case(String[] str, int index, String[]
2  groupList, int timer, String object, String action, float dosage,
3  int reminderInterval){
4      //Transfer to SCCharts language
5      output += ...
6  }
```

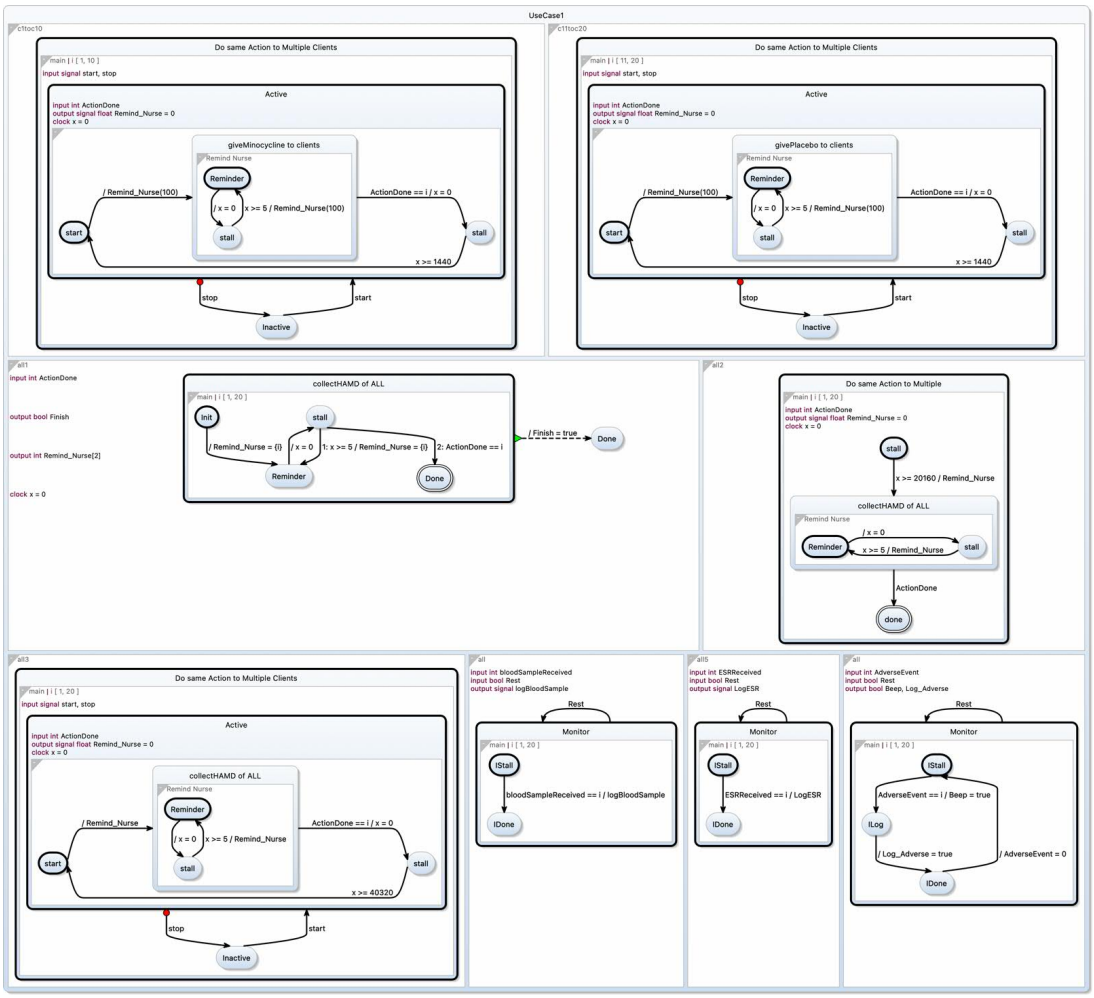


Figure 4.2: Parser Generation Example in Kieler

In this section, we explore the parser mechanism of ISPE-RCT, which extracts data on population, randomization, groups, actions, and customization. An every-case example of the parser function is attached above. As observed, the parameters contain the initial program, 'groupList', timer, object, action, dosage, and 'reminderInterval'. The parser will directly pass the description data to the web service, as using SCCharts to handle strings is complicated. The description data contains how long this clinical trial is, the group allocation, the staff list, the participant list, and the action set. So, the parser functions write the model into the SCCharts programming language and save some digital data in the models, such as the reminder interval, dosage, and the number of participants.

Significantly, only the action logic is of consequence during the parser's operation. The web service can amalgamate the model and stored data, dispatching reminders to participants or the clinical team accordingly. Using the appropriate model function, ISPE-RCT ultimately produces a program based on SCCharts syntax. In subsequent discussions, our focus will exclusively be on SCCharts models, given the SCCharts language is not entirely elaborated upon in this section.

Figure 4.2 offers a comprehensive representation of the model generated by an ISPE-RCT syntax program in Kieler. Upon the commencement of a trial, ISPE-RCT formulates a model for every action, executing them in synchrony. Each model in Figure 4.2 illustrates the state chart of an action. The ensuing portions of this paper elucidate the functioning of each of these models. The top pair and the bottom-left models pertain to the every-case action models as shown in Figure 4.4. The two models in the middle are an after-case example 4.7. The other three models are basic action examples 4.5. The following sections will introduce these models.

4.3.3 Implementation

Until now, ISPE-RCT can parse the user's program to an SCCharts program, which can be simulated in the Kieler. But how do users react to the program?

Action Simulation in Java

Establishing communication between various users begins with the execution of this program in Java. Nonetheless, Java cannot inherently identify and execute a program written in SCCharts syntax. This limitation is aptly addressed by Kieler, which can compile SCCharts programs into Java programs [94], [95]. Drawing inspiration from Kieler, we crafted a bash file that seamlessly integrates the operations of the Kieler compiler and the parser. As a result, ISPE-RCT can effortlessly complete the entire parsing process with the execution of this singular bash file. Finally, it generates two Java files, including the variables and the model logic. That is the way ISPE-RCT parses high-level language to executable SCCharts models.

The web service is designed to run the SCCharts models developed in Java. However, simulating these models directly using Java poses challenges. The executable Java code combines the tick logic, the state creation, the restrictions creation, and a bufferReader between the user and the system. Consequently, executing the models in Java fails to produce the state chart and the associated simulation tool. It is noteworthy that the internal clock is set to tick every minute. This necessitates running the tick logic function and manually entering any state alterations at the input terminal, leading to repetitive actions in Java. To address these challenges, the subsequent sections will explore simulating these models using Kieler.

4.4 SCCharts Part

4.4.1 Overview

Leveraging its capabilities as a synchronous programming language, SCCharts executes models synchronously and employs state charts to depict transitions between states. Kieler serves as a robust tool, illustrating these state changes graphically. Therefore, for the purpose of our study, we've incorporated Kieler to emulate the actions typically observed in clinical trials. Users can conduct these simulations manually, setting user values and advancing to the subsequent state by pressing the 'right arrow' key on their keyboard. This individual action of progressing to the next state by pressing the key is referred to as a 'tick' throughout subsequent sections.

While the specific code and syntax of SCCharts are not the primary focus of this study, the essence lies in understanding the SCCharts models and their role in simulating the progression of actions within a clinical trial.

4.4.2 Actions with the participation of nurses

After Case

```
1    After 1 day giveDose to c1 with 3 mg
```

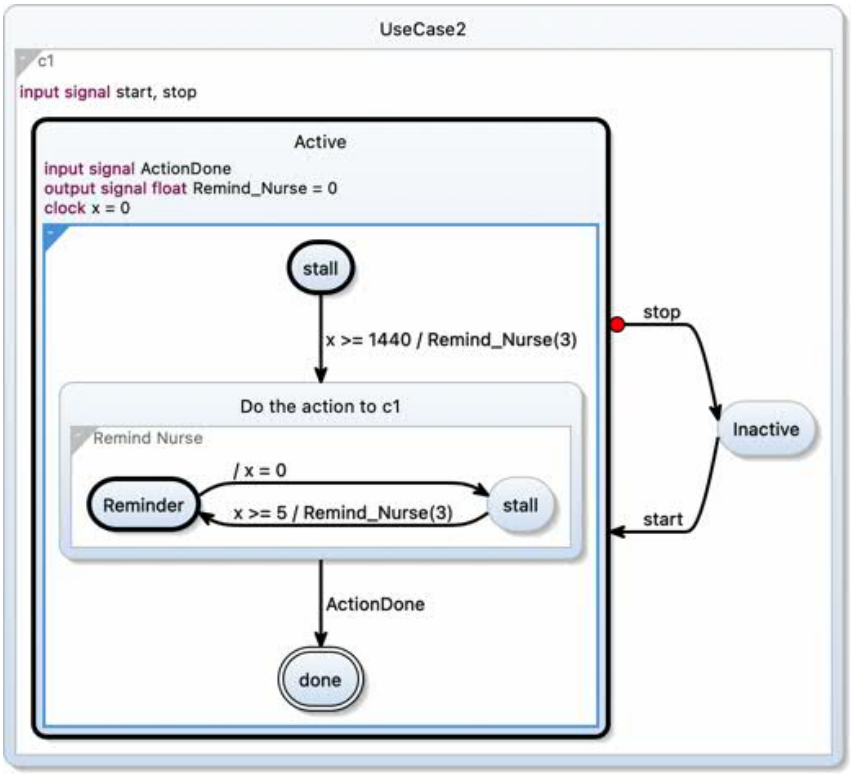


Figure 4.3: Example Model of After Case

The after-case is a simple case. It starts with a 'stall' state, waiting for one day and then moving to the 'Do the action to c1' state. This state has the same logic as the one in the 'every' case. Once it receives an 'ActionDone' signal, this model will move to the final done state. Similarly, this model also has start and stop signals to control this model. Figure 4.3 is a model of the 'after' case:

Every Case

```
1    Every 1 day giveDose to c2 with 3 mg
```

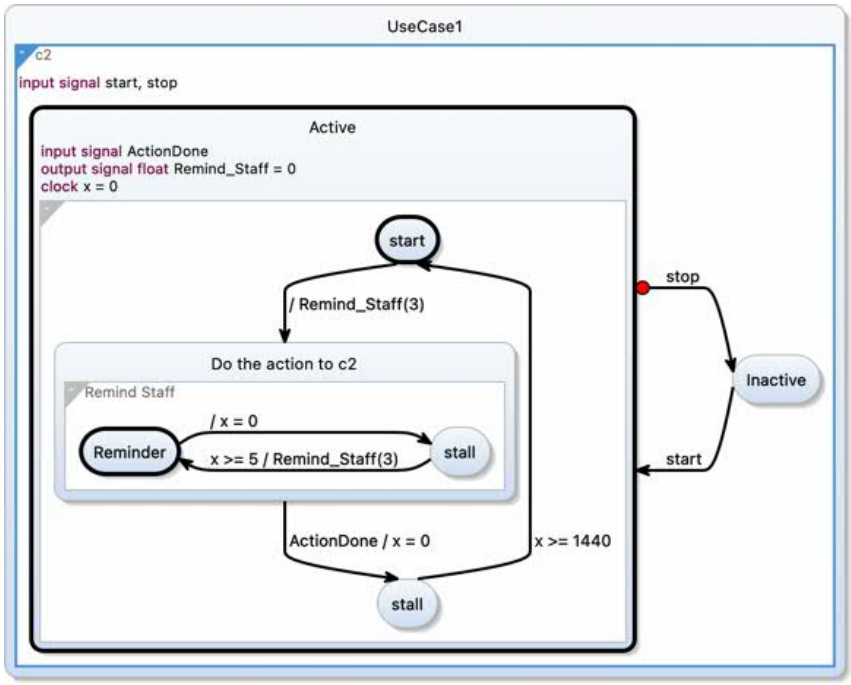


Figure 4.4: Example Model of Every Case

As we can see in Figure 4.4, the largest region, labelled 'UserCase1,' has two input signals: start and stop. They are the signals to manage the running status of this entire model. In the 'Active' region, it initializes the input and output signals and a clock.

SCCharts operates with an inherent clock mechanism. Users can initialize this clock using the 'clock' keyword. To facilitate the manual operation of the model, users can define the delta time for each tick. For instance, by setting the delta time to one, each tick can be interpreted as the passage of one minute. Building on this concept, the model initiates by dispatching a reminder to the staff, subsequently transitioning to the 'loop' region, specifically to the action labelled 'Do the action to c2'.

The 'Do the action to c2' is the core region of the reminder. It will remind the staff every 5 seconds to give some drug to the 'c2' participant 3 mg until the model receives an input signal 'ActionDone' from the staff. As shown in Figure 4.4, the clock x is initialized as 0 and moves to the 'stall' state. The statement $x >= 5/Remind\_Staff(3)$ means

that if the clock x is greater or equal to 5, the system will output the *Remind_Staff* signal as a float number. Once the model receives an 'ActionDone' signal, it will reach the outer 'stall' state and wait for 1440 minutes (1 day) there. That is, after a day, the reminder region will restart.

The floating-point value dispatched as a reminder message can be interpreted as the dosage amount. Additionally, there exists a state termed 'Inactive'. This is because the model is instantiated right at the onset when the parser begins its operation. ISPE-RCT employs start or stop signals to either activate or terminate this model, ensuring that all models remain reusable.

4.4.3 Actions without the participation of nurses

```
1    checkBloodPressure for c1
```

Figure 4.5 is a general model of basic actions.

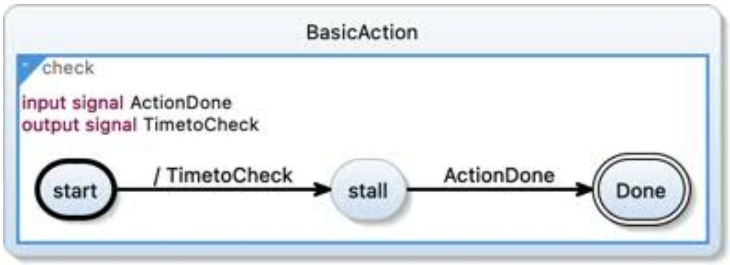


Figure 4.5: Example Model of Basic

It starts from the 'start' state and sends an output signal to the Web Service. Once the system or the staff finishes the action, an 'ActionDone' input signal will be sent to this model. In this case, this basic action is completed.

4.4.4 Actions of Multiple Participants

This section introduces the model of the adverse event first, as a clinical trial usually needs to monitor all the participants simultaneously.

Model of Adverse Event

Figure 4.6 is a model of the 'Adverse Event.' A corresponding code is as follows:

```
1    monitor adverseEvent for ALL
```

This is an example of a 30 30-participant clinical trial. The inner state labelled 'Monitor' can monitor these 30 participants concurrently. The model also starts at the stall state. If one of the participants, c1 raised an adverse event, an integer input would be passed into this model. This integer stands for the participant at this index. Thus,

if this model gets an integer between 1 and 30, ISPE-RCT will emit a beeping sound to alert the staff that one or some participants have an adverse event by checking the value of Beep. The beeping sound is warned.

Meanwhile, ISPE-RCT records any adverse events specific to a participant. Thus, once the beeping concludes, this model will produce a Boolean value, signifying the successful logging of the adverse event.

At last, this model will return to the stall state, poised to address any subsequent adverse events that may arise. Given that SCCharts can execute a transaction with precision and brevity, this model can essentially be viewed as an aggregation of thirty distinct models. Such a compact representation underscores the primary advantage of synchronous programming languages.

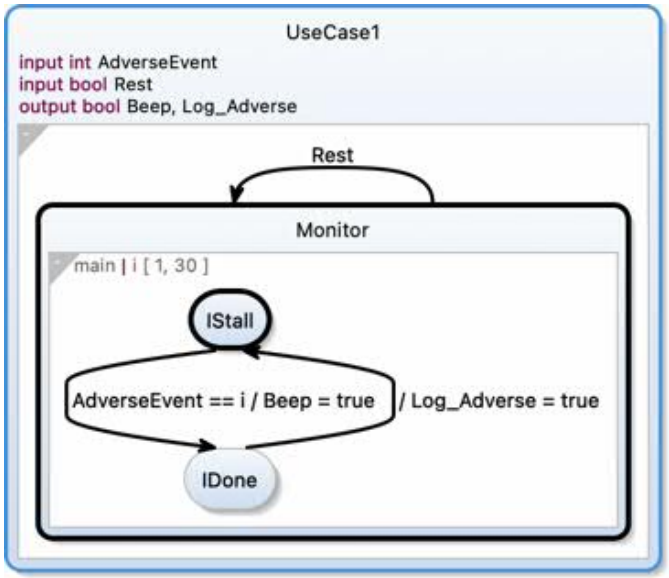


Figure 4.6: Example Model of Adverse Event

Model of After Case for Multiple Participants

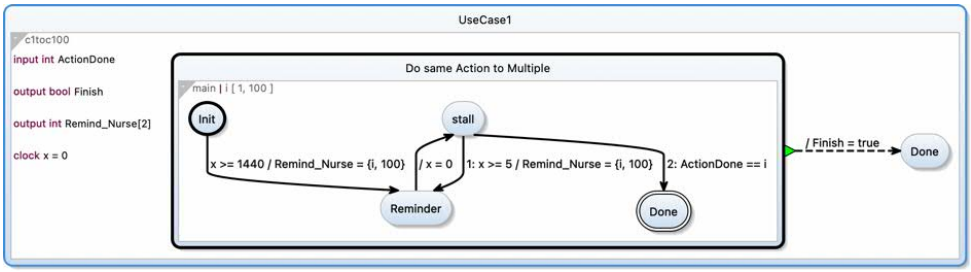


Figure 4.7: Example Model of After Case for Multiple Participants

We introduced the 'After Case' model above, which was only for a specific participant. However, what should the model be like if the user needs an alert action for multiple

participants as a part of the program? Figure 4.7 is an example model with the following design:

```
After 1 day giveDose to clients from c1 to c100
```

This model is essentially the same as the previous one but updated to handle multiple participants. Also, the logic and implementation method is similar to the adverse event model. Consequently, this section will not explain it in detail. Moreover, 'every case' also inherits this logic and method. Thus, this is the way ISPE-RCT handles the same action to multiple users.

4.4.5 Multi-choice Actions

Assume the initial dosage is 2mg and we are collecting questionnaires and modifying the dosage based on the score from questionnaires. When the score is less than 5, we want to keep the current dosage; When the score is greater than 5 and less than 20, we want to decrease the dosage to 1mg. When the score exceeds 20, we want to increase the dosage to 3mg.

```
If collectQuestionnaire < 5, then keep current dosage;
ElseIf 5 <= collectQuestionnaire < 20, then decrease to 1 mg;
ElseIf collectQuestionnaire >= 20, then increase to 3mg
```

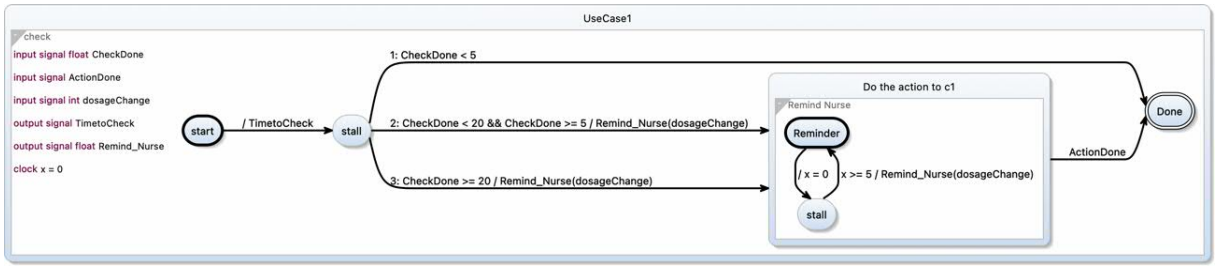


Figure 4.8: Example Model of Multi-choice Actions

Figure 4.8 gives multiple choices according to different values of the check action. The corresponding ISPE-RCT expression is:

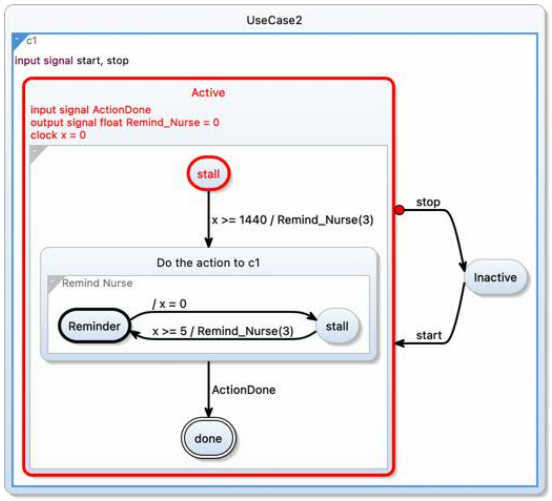
This model begins with a basic action. By getting the value from the basic action, this model can determine the dosage change for the participant. The parser from the users' design generates the input integer, 'dosageChange'. When the 'dosageChange' is greater than 0, it represents the increase in dosage, whereas the negative number stands for the decrease in dosage. The top state movement is to keep the current dosage. The second state movement is decreased to 1mg. Otherwise, increase the dosage to 3 mg. Clinical trials may not frequently use this model, but it is necessary for us to have a primary idea of how this expression is in ISPE-RCT.

4.4.6 Simulation in SCCharts

This section shows the simulation of the model in Section 4.4.2 step by step. Above all, red borders show the current state or region, whereas blue represents the current transaction. Moreover, each SCCharts model must declare an initial and final state.

Initial State

Figure 4.9 is located at the initial state, and all variable values are default. Thus, all the float or integer variables are zero, and the Boolean variables are false. The initial state has a bolded border, whereas the final state has a double-line border. Since there's no input signal at the initial state, we can find the ActionDone, start, stop, and Remind_Nurse signals are all false.



IO	Variable	Value	User Value	History
	_UseCase2_local_ActionDone	false		false
	_region0_c1_start	false		false
	_region0_c1_stop	false		false
	deltaT	0.0000		0.0000
	_UseCase2_local_Remind_Nurse	false	---	false
	_UseCase2_local_Remind_Nurse_val	0	---	0
	sleepT	1000.0000	---	1000.0000

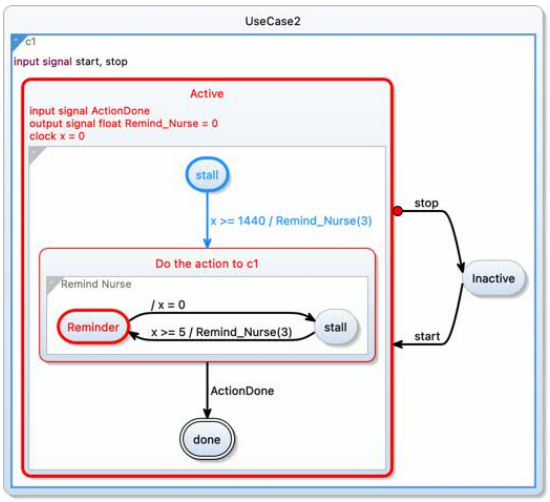
(a) Initial State (b) Initial State Vaules

Figure 4.9: Simulation of The Initial State

Reminder State

Figure 4.10(a) indicates the transaction from the initial state to the reminder region. To simplify the process, the delta-T value is 1440, so one tick means 1440 minutes passed. Regarding the history in figure 4.10(b), we can find the top three lines are remaining false. We change the delta-T to 1440 to satisfy the requirements of this model. Then, at row 5, the Reminder_Nurse signal changes to true, and its output integer value at row 6 is 3.

Since x equals 1440 at this stage, it moves to the reminder state and sends a reminder with 3 mg as the message. ISPE-RCT checks both the float and Boolean values of Reminder_Nurse to get an entire reminder message.



(a) Reminder State

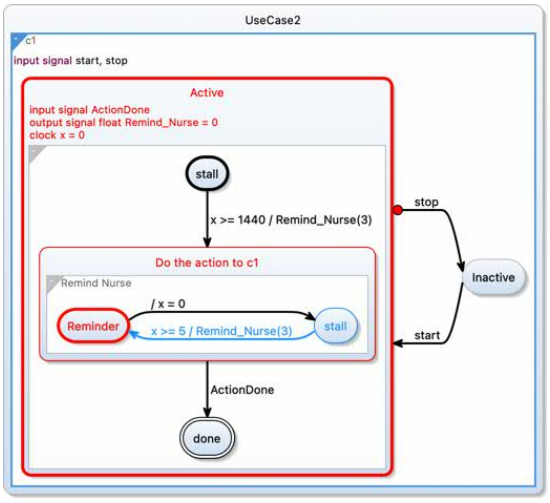
IO	Variable	Value	User Value	History
	_UseCase2_local_ActionDone	false		false, false
	_region0_c1_start	false		false, false
	_region0_c1_stop	false		false, false
	deltaT	1440.0000		1440.0000, 0.0000
	_UseCase2_local_Remind_Nurse	true	---	true, false
	_UseCase2_local_Remind_Nurse_val	3	---	3, 0
	sleepT	1000.0000	---	1000.0000, 1000.0000

(b) Reminder State Values

Figure 4.10: Simulation of The Second State

Reminder Region

Figure 4.11(a) shows the transaction in the 'Do the action to c1' region. Regarding the history in Figure 4.11(b), the top three rows remain false. It took five one-minute ticks before achieving the current transaction from the inner stall stage to the reminder state. That is showing as 'true, false, false, false, false, false, true'. Once it reaches the reminder state, this model will immediately jump to the inner stall state and reset the clock to zero to restart the time counting. There are 5 ticks between every two reminder signal outputs. Benefiting the internal clock, SCCharts manages reminder outputs accurately.



(a) Reminder Region

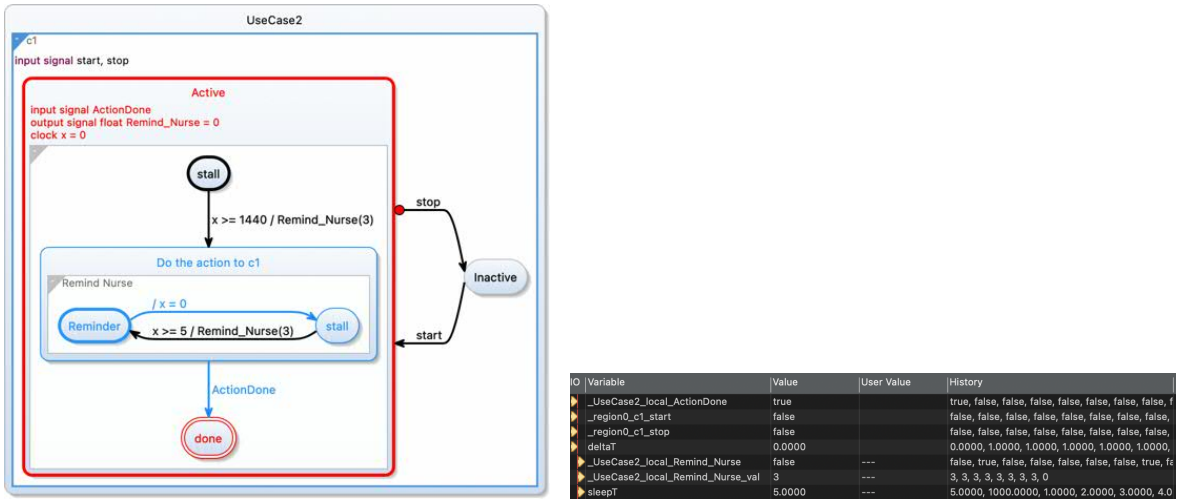
_UseCase2_local_ActionDone	false		false, false, false, false, false, false, false, false
_region0_c1_start	false		false, false, false, false, false, false, false, false
_region0_c1_stop	false		false, false, false, false, false, false, false, false
deltaT	1.0000		1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 0.0000,
_UseCase2_local_Remind_Nurse	true	---	true, false, false, false, false, false, true, false
_UseCase2_local_Remind_Nurse_val	3	---	3, 3, 3, 3, 3, 3, 3, 0
sleepT	1000.0000	---	1000.0000, 1.0000, 2.0000, 3.0000, 4.0000, 5.0

(b) Reminder Region Values

Figure 4.11: Simulation of The Reminder Region

Done State

The system needs an 'ActionDone' signal to reach the final state. In this manual simulation, the user is supposed to set the value of 'ActioinDone' as true, representing the model has received the signal. Regarding the history in figure 4.12(b), we input the 'ActionDone' value as true, and the model will move to the final done state. Besides, blue arrows and states in Figure 4.12(a) show the transaction from the reminder region to the final state. Here comes the end of this model. The internal ticks can generate output signals and the input signals need to be completed by the clinical team.



(a) Done State (b) Done State Values

Figure 4.12: Simulation of The Fourth State

4.5 Web Service

This section will introduce the features of web service and a simulation with a simple program. In addition, this section will also display the UI, data storage, data retrieval, and interaction between different users in detail.

4.5.1 Grab Data from Initial Program

As mentioned above in 4.3.2, saving data in a SCCharts model is complicated. Thus, only some essential data is saved in the SCCharts models. The server can retrieve those digital numbers, for example, the dosage and reminder interval.

However, other information should be retrieved from the initial program. Section 4.3.1 displayed those functions and the method to save data. Therefore, the server can grab data by importing the parser and calling functions. Specifically, the server should get the *populatioinList, groupList* and map them to the *actionList*.

4.5.2 Simulation on the Web Service

Assume we have a scenario in which Participant 1 needs an intervention every 5 seconds, whereas Participant 2 needs an intervention every 6 seconds. The following shows a simulation of the tiny program.

```
1 //Clinical Trial - 1
2 ...
3 every 5 second giveDose to p1
4 every 6 second giveDose to p2
5 ...
```

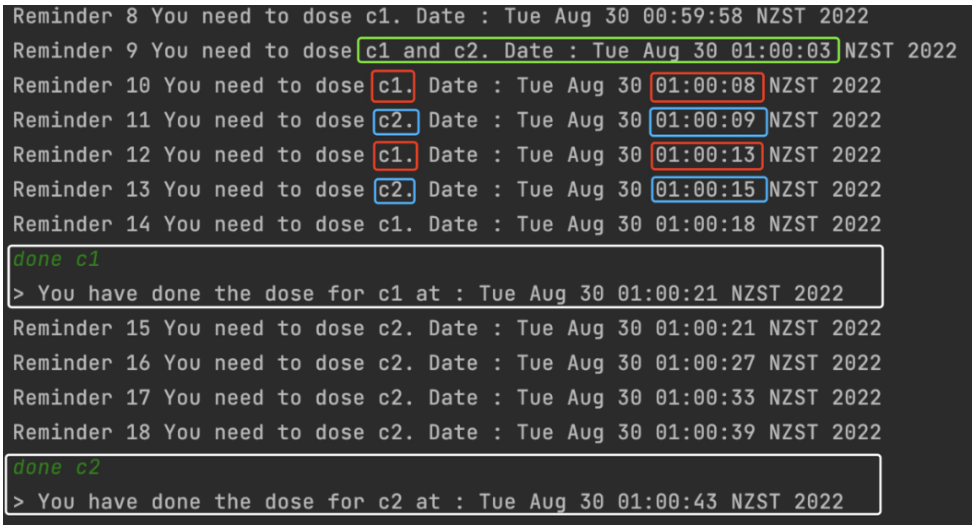


Figure 4.13: Simulartion of the Tiny Program

To show the simulation more straightforwardly, there is the message list of the staff in Figure 4.13. What's more, the first action is set as sending the reminder message every 5 seconds, whereas the second one sends the reminder message every 6 seconds. As you can see, reminder 9 asks the staff to dose participants c1 and c2. Also, the following five reminder messages' time stamp follows the alert rules. Once this staff sends a message of 'done c1', ISPE-RCT will set the 'actionDone' signal of c1 to be true, so there is no more reminder for c1 but a message to ensure that the system acknowledges the dose for c1 has been done. At last, there is a confirmation that action c2 has been done.

4.5.3 User Interface and Features of Web Service

To elucidate the web service, this paper commences with exploring the user interface (UI) and the communication mechanisms employed between various users. The foundation of the web service is the Transmission Control Protocol (TCP). Consequently, the server and client directories house a connection file, a user details file, and dedicated sending and receiving threads.

Server UI

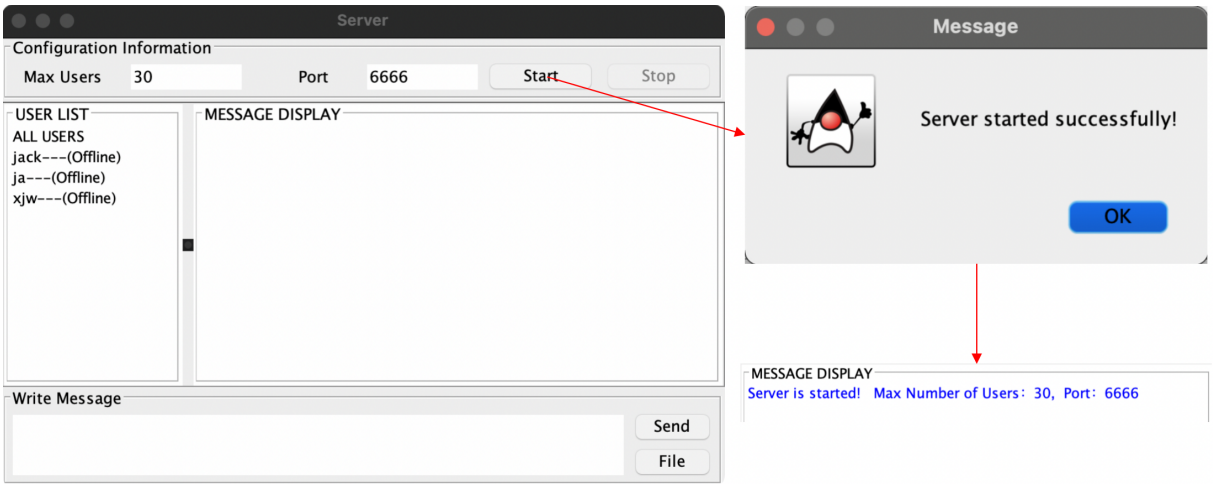


Figure 4.14: User Interface of Server

Figure 4.14 showcases the user interface (UI) of the server platform. The total of participants and staff determines the initial maximum user count. The designer retains the flexibility to modify this value at any time directly from the server platform. Additionally, the designer can broadcast messages to all user groups. As depicted in Figure 4.14, upon activating the start button, the server commences its operation on port 6666. Users can send and receive messages, the content of which is visible in the designated message display area. Given that this represents the server's initial page, all users listed on the left sidebar are currently offline.

Moreover, the server connects MySQL as its database to record all the users' data. The database is also serving on port 6666.

Client UI and Features

The client-side design mirrors that of the server side. Both the clinical team and participants utilize the client side for mutual communication. For instance, users can transmit files when the team requires participants to submit questionnaires. Conversely, when the clinical team needs to relay input signals to the SCCharts models, they can dispatch a message to the server. The server software can authenticate this message and subsequently modify the signal values. The following sections will elucidate the communication dynamics between users and the server.

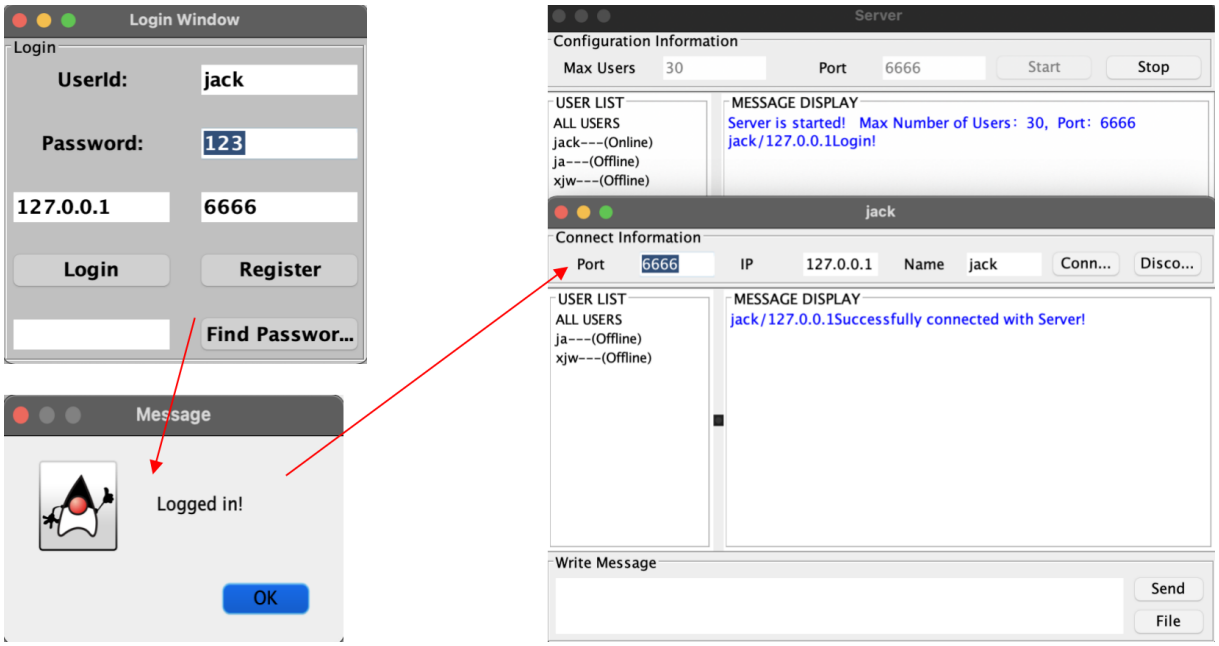


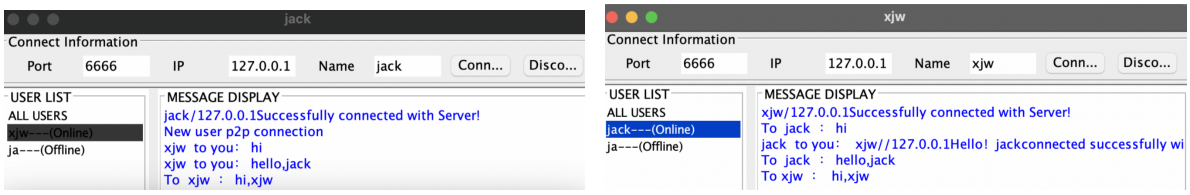
Figure 4.15: User Interface of Client

Figure 4.15 is the UI of the client platform. The clients' login window is on the top left. Users can either register a new account or log in using an existing account. Moreover, users can click the 'Find Password' button if they forget their password. The server will send the user's password to his email address.

The default values of IP and port are set as the same as those of the server. Once a user logs in to the client platform, a message will be displayed on the server side, and the user's page will show the login details, including the username and IP address. Also, on the server and other users' side, the status of this user, Jack, will be online. At this stage, user Jack can receive and send messages to the server and all other users.

Communication

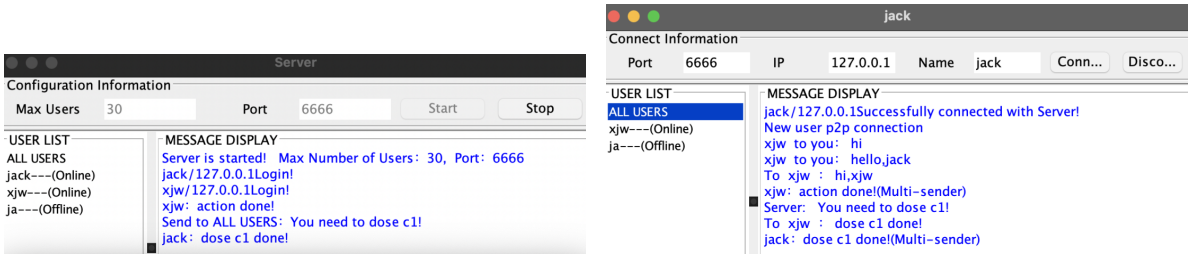
Figure 4.16 shows the communication between different users. For example, as the user 'xjw,' I can find two users from my user list that I can contact. To send a message to Jack, xjw can click his name from the left sidebar, type in a message, and click send. The message display on both users are shown in Figure 4.16(a) and Figure 4.16(b).



(a) User 1 (b) User 2

Figure 4.16: Communication between Users

Figure 4.17 shows user and server communication. The 'ALL USERS' has different logic on the server and client sides. The server can send messages to all users by sending them to the 'ALL USERS.' However, regarding the needs of the user story, if a user sends messages to the 'ALL USERS,' it is equivalent to sending them to the server.



(a) Server Sends Messages to All Users (b) User Sends Messages to the Server

Figure 4.17: Communication between Users and Server

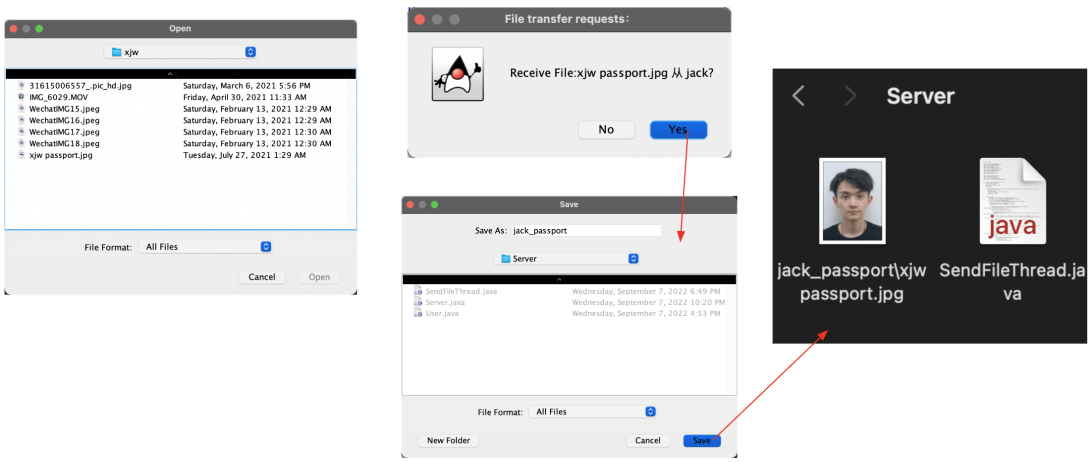


Figure 4.18: File Sending and Receiving

Figure 4.18 illustrates sending and receiving a picture. Since a clinical trial team usually needs to collect files from all participants, this is a simulation of collecting the participants' passports. The picture at the left shows the UI after a user clicks the 'File' button in the chat with the Server. Thus, a popup window comes to the server side, and the server can save this picture. The last picture shows the server received this picture and renamed it.

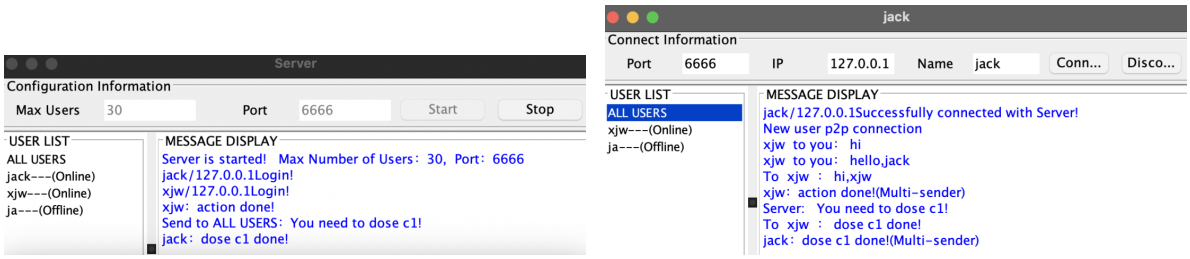
4.5.4 Data

Figure 4.19 shows the users' details record. The 'id' column matches the label of participants in the parser. The message column is the place to record any messages that are sent to the offline user. When the user logs in to the client side again, the platform will automatically read the messages from this column and send them to this user. Figure 4.20(a)

displays the server leaving two messages to Jack, and Figure 4.20(b) demonstrates Jack receiving messages at the next login. This is the feature to avoid any missing messages.

	* Id int	username varchar(1(	password varchar(1(	youxiang varchar(100)	message varchar(1(	state int	serverPort int	idAdres varchar(1(
1	1	jack	123	jiawangxing@gmail.com	Server left a	0	0	127.0.0.1
2	2	ja	ja	ja@gmail.com		0	0	127.0.0.1
3	3	xjw	xjw	xjw@gmail.com		0	0	127.0.0.1

Figure 4.19: Datebase of Participants



(a) Server Sends Messages to Offline User (b) User Receive Messages at the Next Login

Figure 4.20: Process of Leaving Messages

4.6 Summary

This chapter delves into the development and operational intricacies of the ISPE-RCT system, presenting a pioneering approach to synchronized clinical trials through a bespoke high-level language. We initiate with an in-depth exploration of the ISPE-RCT syntax, followed by its transformation into an executable Java program harmoniously integrated with SCCharts models.

The web service's architecture is elucidated, emphasising the nuanced interplay between server and client. This includes mechanisms for data acquisition, particularly the pivotal management of digital data, such as dosage specifications and reminder intervals. A simulation within this section serves as a tangible demonstration of the system's capabilities.

The user interface (UI) is introduced on the server side, detailing user management and MySQL connectivity features. The client side receives equal attention, exploring its user interface, login procedures, and communication paradigms encompassing user-to-user and user-to-server interactions. The file transfer mechanism, facilitating the smooth exchange of essential documents like passports in a simulated clinical trial, is meticulously explained. A segment deserving special mention addresses the management of offline messages, detailing the methodology for storing and retrieving messages from the database, ensuring no critical information is lost. This approach marries the benefits of synchronous

programming languages with contemporary web service paradigms, presenting a platform poised to revolutionize clinical trial efficiency, security, and innovation.

Users must design an ISPE-RCT syntax program. ISPE-RCT is adept at interpreting this high-level language program, parsing it into an executable Java program integrated with SCCharts models. Ultimately, the ISPE-RCT web service executes this Java program, enabling users to log into the service and participate in the clinical trial. This process exemplifies how ISPE-RCT harnesses the strengths of synchronous programming languages, applying them to clinical trial procedures and executing them synchronously within the web service.

5

Research Tests

At the inception of this study, ISPE-RCT was conceptualized for a clinical trial based in New Zealand, targeting depression [50]. This trial, designed by Robin J. Murphy et al., sought to investigate the effects and potential side effects of regular microdosing of psychedelics. This clinical trial can reshape treatment paradigms, with subsequent clinical studies possibly exploring microdosing psychedelics as either a primary intervention or an adjunct to psychotherapy. Such treatments could address various conditions, including depression, addiction, eating disorders, obsessive-compulsive disorders, and even palliative care. The trial is characterized by its randomized, double-masked, and placebo-controlled design. The capabilities of ISPE-RCT to facilitate randomized and double-masked features are elaborated upon in Chapter 3.1. This chapter will demonstrate the application of these features in the context of several concluded clinical trials.

Microdosing [96], [97] involves the consistent administration of sub-threshold doses of psychedelics, such as lysergic acid diethylamide (LSD) or psilocybin. T Lea et al. [98] highlighted potential challenges associated with microdosing, including focus difficulties, sensations of being overwhelmed, hypersensitivity, sleep disturbances, intense euphoria, and experiences reminiscent of a complete psychedelic trip. Given the potential risks of psychedelics, monitoring each participant is paramount. The original design proposed family members as the primary monitors. However, as the clinical trial is designed for at least six weeks, humans can get distracted by other things and forget or miss monitoring a participant's status. That's why a reliable monitor and alert system will be much more suitable for them.

A recurring aspect of this study involves administering microdoses to participants and collecting their feedback through questionnaires. The SCCharts models within ISPE-RCT were crafted to facilitate these actions. For some reason, ISPE-RCT wasn't deployed in Robin's clinical trial.

This chapter aims to assess these models' feasibility, strengths, and limitations by applying them to completed clinical trials. We will also evaluate system performance metrics like memory and CPU usage. While ISPE-RCT's initial design was rooted in depression-focused clinical trials, the examples presented here span beyond depression, even touching upon a contemporary concern: COVID-19. These examples offer a diverse look at clinical trial phases, intervention techniques, group structures, and analytical approaches. In terms of randomization and group design within ISPE-RCT, this chapter categorizes clinical trial examples based on their intervention models and participant group numbers. Notably, all referenced clinical trials were concluded within the past decade.

Due to the extensive design details of some clinical trials, we've relegated comprehensive introductions and specific study designs to Appendix 1 for the reader's convenience.

5.1 Single Group Assignment

Although the primary objective of this study was to craft a programming environment tailored for RCTs, our inability to deploy ISPE-RCT in Robin J. Murphy's clinical trial prompted us to evaluate its merits and limitations. We aimed to juxtapose ISPE-RCT's existing capabilities against the diverse requirements of various clinical trials to understand its strengths and areas for improvement. Clinical trials with a single group are the simplest, and thus they form the initial section of this research process. Typically, single-group clinical trials are open-labeled.

5.1.1 iTBS for Adolescent Depression

The University of North Carolina, Chapel Hill, held a Phase I clinical trial titled Intermittent Theta Burst Transcranial Magnetic Stimulation (iTBS) for Adolescent Depression [99].

Study Design

This study is a treatment-oriented, open-label clinical trial employing a single-group assignment model with no distinct allocation. The intervention under investigation is iTBS therapy, administered to adolescent participants diagnosed with depression.

In the experimental arm of this study, participants undergo iTBS therapy delivered via a specific Transcranial Magnetic Stimulation (TMS) protocol. This protocol involves electromagnetic stimulation to target areas of the brain.

The iTBS treatment process involves two key devices: a Motor Threshold Coil and a Treatment Coil. Before commencing treatment, a Motor Threshold determination process identifies the intensity and location of the iTBS treatments. This is achieved by applying an escalating magnetic field intensity, stimulating the brain's motor region. Observable thumb movement serves as the indicator for treatment intensity, while the sensory area parallel to this location is identified for treatment.

The iTBS Treatment Coil administers the actual therapy. This coil utilizes a specific TMS protocol to deliver a magnetic field in triplet bursts (three rapid stimulations at 50 Hz frequency), with a repetition rate of 5 Hz for 2 seconds (30 pulses). Each treatment lasts roughly 3 minutes, consisting of 2 seconds of active treatment followed by 8 seconds of rest, and this cycle is repeated 20 times, totalling 600 pulses. Treatment sessions are conducted 20 times, from Monday to Friday, over four consecutive weeks.

Outcome Measure

Similar to many clinical trials in the field of depression, the outcome of this experiment is also determined through survey completion by the participants. The required data is collected, and the results are analyzed based on the responses provided in these questionnaires. Table 5.2 records the questionnaires used and their corresponding collection times.

Outcome Measure	Time Frame
HAM-D	Baseline, Week 1, Week 2, Week 3, Week 4
CDRS-R	Baseline, Week 1, Week 2, Week 3, Week 4
NSSIB	Baseline, Week 1, Week 2, Week 3, Week 4
C-SSRS	Baseline, Week 1, Week 2, Week 3, Week 4 Week 5, Week 8, Week 16

Table 5.1: Questionnaires used in iTBS clinical trial

Application

Regarding the currently available syntax in ISPE-RCT, it is not convenient to express and execute the actions with a device. Therefore, the program below only involves the actions of collecting these questionnaires.

```
1 //A Program for iTBS Research
2 //Population Expression
3 Participant 5
```

```
Staff 3
Group 1
//Customization Expression
Interval Time 4 weeks
Alert 5
AlertActions collectHAMD collectCDRSR collectNSSIB collectCSSRS
//AlertAction Expression
every 1 week collectHAMD for ALL
every 1 week collectCDRSR for ALL
every 1 week collectNSSIB for ALL
every 1 week collectCSSRS for ALL
after 5 weeks collectCSSRS for ALL
after 8 weeks collectCSSRS for ALL
after 16 weeks collectCSSRS for ALL
stop
```

5.1.2 The Feasibility Engage Therapy With Video Support for Homebound Older Adults

A study for homebound older adults [100], [101] started on May 2022 by the Weill Medical College of Cornell University. This study was located at Weill Cornell Medicine and only had 8 participants.

Study Design

The study design at hand revolves around treating depression through a unique approach that integrates behavioural talk therapy and technology. The study follows a single-group assignment model with no allocation or masking—making it an open-label intervention.

The participant group involved is exposed to "Engage Prism 2.0," a 9-week program that merges Engage and guided usage of Prism technology.

This study's intervention or treatment method is labelled as "Engage Prism (2.0)." It represents a concise behavioural treatment for depression (Engage) and a tablet-based social support system (Prism 2.0). By combining these two elements—behavioural therapy and technology—the program aims to combat depression more holistically and multifacetedly.

Outcome Measure

The outcome measure includes the Client Satisfaction Questionnaire (CSQ) and the 24-item Hamilton Depression Rating Scale(HAM-D). The CSQ is a three-item questionnaire used to measure client satisfaction with therapy.

Outcome Measure	Time Frame
HAM-D	Baseline, Week 9, Week 12
CSQ	Week 6, Week 12

Table 5.2: Questionnaires used in "Engage Prism 2.0" clinical trial

Application

```
//A Program for Engage Prism 2.0 Research
//Population Expression
Participant 8
Staff 3
Group 1
//Customization Expression
Interval Time 9 weeks
Alert 5
AlertActions collectHAMD collectCSQ
//AlertAction Expression
collectHAMD for ALL
after 9 weeks collectHAMD for ALL
after 12 weeks collectHAMD for ALL
after 12 weeks collectCSQ for ALL
stop
```

5.1.3 Summary of Single Group Assignment

Single-group clinical trials, or single-arm trials, are a type of clinical study where all participants receive the same intervention instead of RCTs.

Single-group clinical trials are commonly used in Phase I and Phase II trials for testing a new drug or treatment's safety, tolerability, and dosage. These are essential steps before moving on to larger RCTs. Many single-group assignments involve a device or software, as it is not necessary to design an RCT to investigate the effectiveness of a device. Some rare disease studies also use single-group clinical trials because recruiting enough participants for an RCT might be challenging. Similarly, for specific populations, such as the elderly, children, or patients with specific genetic mutations, where it may be challenging to

conduct a randomized controlled trial due to ethical or practical issues, single-arm trials can play an important role.

Moreover, one of the characteristics of the single-group clinical trial is simplicity. The programs above, however, have too many duplicate expressions, which wastes memory and time. For example, collecting different questionnaires in the first clinical trials has the same period. It will greatly improve if ISPE-RCT integrates and handles these actions with the same type and period. Additionally, since the design of ISPE-RCT was based on the interaction between different user groups, it did not support a function for connecting any device or software. Furthermore, ISPE-RCT doesn't have a suitable SCCharts model for an action repeated every Monday to Friday. Section 6.1 will discuss the performance of SCCharts models in depth.

5.2 Parallel Assignment - Two Groups

Among all the clinical trials listed in the National Library of Medicine website [102], more than half of parallel-assignment type clinical trials have two participant groups. Most of these clinical trials have a placebo group to prove the effectiveness of the new treatment. This section introduces two clinical trials in the COVID-19 area. One has the controlled group, whereas the other one doesn't set the controlled group.

5.2.1 COVID-19 Study of Repurposed Medications - Arm C (Fluticasone)

Susanna Naggie et al. completed a study evaluating the effectiveness of Fluticasone in May 2022. [103]

Study Design

This study has two groups, an experimental group, and a placebo comparator group. The research team wants to give the participants 200 µg an intervention or placebo once daily for 14 days.

Outcome Measure

This study used the Patient Reported Outcomes Measurement Information System 29-item Health Profile(PROMIS-29) and a self-designed COVID Clinical Progression Scale to measure the participant's physical and mental condition in various ways. The PROMIS-29 consists of seven health domains, each with four 5-point items and a pain intensity score on a numerical scale from 0 to 10. The seven health domains include physical function, fatigue, pain interference, depressive symptoms, anxiety, the ability to participate in social

roles and activities, and sleep disturbance. The raw score ranges from 4-20, with higher scores correlating with better physical function outcomes.

A score from 0 to 8 on the COVID Clinical Progression Scale represents no infection to death. Higher numbers indicate that the participant was in poorer health.

Apart from the scales, this study also need to record the time to sustained recovery, the number of death, the number of participants with mortality and any hospitalization or adverse events.

Outcome Measure	Time Frame
COVID Clinical Progression Scale	Day 7,14,28
PROMIS-29 - Physical Function PROMIS-29 - Fatigue PROMIS-29 - Pain PROMIS-29 - Depression PROMIS-29 - Anxiety PROMIS-29 - Social PROMIS-29 - Sleep	Day 7,14,28, and 90
Time to Sustained Recovery Number of Participants With Death Number of Participants With Mortality Number of Participants With Urgent Care, Emergency Room Visit, or Adverse Event	Up to 28 days

Table 5.3: Outcome Measure in COVID-19: Fluticasone Study

Application

```
1  //Program for Fluticasone Research
2  //Population Expression
3  Participant 1407
4  Staff 15
5  Group 2
6  //Customization Expression
7  Interval Time 4 weeks
8  Alert 5
9  BasicActions logDeath logMortality adverseEvent
10 AlertActions giveFluticasone givePlacebo collectCCPS
11 collectPROMIS
12 //AlertAction Expression
13 every 1 day giveFluticasone for group1 with 200 mg
14 every 1 day givePlacebo for group2 with 200 mg
```

```
every 1 week collectCCPS for ALL
every 4 weeks collectPROMIS for ALL
//BasicAction Expression
logDeath for ALL
logMortality for ALL
monitor adverseEvent for ALL
stop
```

5.2.2 Antibody Plasma Research Study in Hospitalized Patients

This study [104], [105] was held by Luther A. Bartelt and David M. Margolis from the University of North Carolina, Chapel Hill.

Study Design

Interventional Model Description: This study has two treatment arms, and participants will be randomly assigned to one arm:

- CCP1 - CCP tested for the presence of anti-SARS-CoV-2 antibodies and assigned high-titer
- CCP2 - CCP tested for the presence of anti-SARS-CoV-2 antibodies and assigned standard-titer

Masking Description: Neither the research team nor the participants will know which treatment (CCP1 or CCP2) the participant will receive.

Outcome Measure

The following table shows the outcome measure of this study.

Application

```
//Program for Antibody Plasma Research
//Population Expression
Participant 56
Staff 5
Group 2
//Customization Expression
Interval Time 4 weeks
Alert 5
BasicActions adverseEvent
```

Participant Group/Arm	Intervention/Treatment
Experimental: High-Titer(CCP1) Participants will be randomized within eight days of the onset of COVID symptoms and no more than 48 hours after hospital admission. They will receive high-titer ABO-compatible convalescent COVID-19 plasma (CCP1) within 24 hours of randomization.	Biological: High-titer Convalescent COVID-19 Plasma(CCP1) At least two units of CCP transfused 4-24 hours apart on Study Day 0. A third unit may be administered, if available.
Active Comparator: Standard-Titer(CCP2) Participants will be randomized within eight days of the onset of COVID symptoms and no more than 48 hours after hospital admission. They will receive standard-titer ABO-compatible convalescent COVID-19 plasma (CCP1) within 24 hours of randomization.	Biological: Standard-titer Convalescent COVID-19 Plasma(CCP2) At least two units of CCP transfused 4-24 hours apart on Study Day 0. A third unit may be administered, if available.

Table 5.4: Arms and Interventions of Antibody Plasma

Outcome Measure	Time Frame
Total Number of Serious Adverse Events	14 days
Median Time to Hospital Discharge Following the First Dose of CCP	Up to 28 days

Table 5.5: Outcome Measure of Antibody Plasma

```
10 AlertActions giveHighTiter giveStandardTiter
11 //AlertAction Expression
12 after 4 hours giveHighTiter with group1
13 after 4 hours giveStandardTiter with group2
14 //BasicAction Expression
15 monitor adverseEvent for ALL
16 stop
```

5.2.3 Summary

The majority of two-group clinical trials are structured as RCTs. Since we have previously highlighted the strengths of ISPE-RCT in the context of RCTs, we will concentrate on its

limitations in this section. Regarding the two clinical trials mentioned earlier, ISPE-RCT exhibits the following limitations:

- The user can use the keyword 'Interval Time' to pre-set the clinical trial duration. For example, 'Interval Time 4 weeks' means this program will terminate after four weeks. However, there sometimes will be some follow-up actions. That means the user has to re-design a new program for the follow-up actions if needed.
- Many clinical trials need to record the data of the total number of a case or an event. For example, Susanna Naggie et al.'s study used the number of participants with death to measure the effectiveness of Fluticasone. However, ISPE-RCT doesn't have a proper SCCharts model for it.
- Regarding the syntax and semantics of ISPE-RCT, there is no proper function to express an action without a specific time. For example, the time interval between the two interventions for participants in this clinical trial can be between 4 and 24 hours.

5.3 Parallel Assignment - Multiple Groups

A study with multiple participant groups or arms has a more complicated study plan. Applying the ISPE-RCT to such clinical trials can better help us detect the usability and applicability of ISPE-RCT in complex clinical trials.

5.3.1 Omega 3 for Treatment of Depression in Patients With Heart Failure (OCEAN)

Wei Jiang led this clinical trial in 2014. [106]–[108]

Study Design

This study has two active comparator groups and one placebo comparator group, and the detailed study design is in Appendix 1.

Outcome Measure

EPA values are mean levels adjusted for age, race, sex, treatment site, and baseline EPA level. Red blood cell (RBC)/plasma EPA values are expressed as a percentage of total fatty acids identified.

Outcome Measure	Time Frame
HAMD	Baseline, Week 12
RBC/Plasma EPA Values	Baseline, Week 12

Table 5.6: Outcome Measure of EPA/DHA

Application

```
//Program for OCEAN Research
//Population Expression
Participant 108
Staff 10
Group 3
//Customization Expression
Interval Time 12 weeks
Alert 5
BasicActions logEPAValue
AlertActions giveEPA/DHA giveHighEPA givePlacebo collectHAMD
//AlertAction Expression
every 1 day giveEPA/DHA with group1
every 1 day giveHighEPA with group2
every 1 day givePlacebo with group3
after 12 weeks collectHAMD for ALL
//BasicAction Expression
logEPAValue for ALL
after 12 weeks logEPAValue for ALL
stop
```

5.3.2 Effectiveness of Methylphenidate in Improving Cognition and Function in Older Adults With Depression

Helen Lavretsky led the study for the effectiveness of Methylphenidate [109]–[111], which started in 2008.

Study Design

This study has three active comparator groups. The detailed study design is also in Appendix 1.

Outcome Measure

The Quality of Life Enjoyment and Satisfaction Questionnaire - Short Form (Q-LES-Q-SF) is a 16-item self-administered questionnaire that measures life satisfaction over the past week. Each item is rated on a 5-point scale from 1 (very poor) to 5 (very good). The total score is given for items 1-14. The minimum raw score for the Q-LES-Q-SF is 14, and the maximum is 70, with higher scores representing better outcomes.

The Clinical Global Impressions (CGI) scale is a widely used measure in mental health, particularly in clinical trials of psychiatric medications, to assess global functioning. Developed by the National Institute of Mental Health in the USA, the scale has been used since the late 1970s.

Outcome Measure	Time Frame
HAMD	Baseline, Week 16
Q-LES-Q-SF	Baseline, Week 16

Table 5.7: Outcome Measure of Citalopram & Methylphenidate

Application

```
//Program for Methylphenidate Research
//Population Expression
Participant 181
Staff 18
Group 3
//Customization Expression
Interval Time 16 weeks
Alert 5
AlertActions giveCitalopram giveMethylphenidate givePlacebo
collectHDRS collectQ-LES-Q-SF collectCGI
//AlertAction Expression
every 1 day giveCitalopram for group1 with 20 mg:
If collectCGI > 2, then increase to 40 mg
Else keep current dosage
every 12 hours givePlacebo for group2 with 40 mg
every 12 hours giveMethylphenidate for group3 with 20 mg
every 12 hours givePlacebo for group4 with 40 mg
every 1 day giveCitalopram for group5 with 20 mg:
If collectCGI > 2, then increase to 40 mg
Else keep current dosage
every 12 hours giveMethylphenidate for group3 with 20 mg
```

```
after 16 weeks collectHDRS for ALL
collectQ-LES-Q-SF for ALL
after 16 weeks collectQ-LES-Q-SF for ALL
stop
```

5.3.3 Summary

The above two clinical trials are complex multiple-group parallel assignments. The first one is similar to a two-group parallel assignment but with three groups. The second one has three arms, each with an intervention group and a control group, to get a conclusion with strong evidence.

Shortcomings:

- ISPE-RCT is not flexible enough for users. It doesn't have a suitable SCCharts model for some complex actions. For example, users must specify the dosage and frequency when designing the model. ISPE-RCT currently only supports users inputting a fixed dose and does not support inputting a dose range.
- It doesn't have a fully acceptable function to handle a group with an internal intervention group and a control group.

5.4 Performance Test

While we apply ISPE-RCT to these six completed clinical trials, we also want to test the performance of ISPE-RCT, including the CPU and memory usage of running ISPE-RCT. Since ISPE-RCT runs based on Java, our first plan was to use OperatingSystemMXBean [112], a public Java interface, to test this Java project's CPU and memory usage. For example, we can use the 'getProcessCpuLoad()' method to return CPU usage for the Java Virtual Machine process. Method 'getFreePhysicalMemorySize()' returns the number of free physical memory in bytes and method 'getTotalPhysicalMemorySize()' returns the total physical memory size. Thus, we can use the 'getFreePhysicalMemorySize()' and 'getTotalPhysicalMemorySize()' methods to return the percentage of loaded memory indirectly. Using MXBean is a straightforward implementation for the performance test.

However, regarding G Kereszty's study [113], we found OperatingSystemMXBean cannot return a very reliable CPU usage. [114]–[116]. Our usage tests will employ the Operating System and Hardware Information (OSHI) [117]. OSHI is an open-source Java library that allows developers to access detailed information about a system's hardware and operating system. This encompasses data on the CPU, memory, hard drives, network interfaces, and battery status, among other specifics. The main advantage of using OSHI is that it provides a platform-independent interface for retrieving such details, making

it easier for developers to collect system statistics across platforms without resorting to native code or platform-specific tools. Given its capabilities, developers can use OSHI in various applications, from system monitoring tools to diagnostic utilities. As an abstraction over native system calls and other methods for retrieving hardware and OS information, OSHI simplifies accessing this data consistently across different systems.

Moreover, regarding the usage tests of these studies [118]–[120], we conclude that the CPU and memory usage data from OSHI is the most accurate. Hence, we'll monitor CPU and memory usage while doing the application on the previous six clinical trials to test the performance of ISPE-RCT. We'll then show the results and discussion in Chapter 6.

6

Results and Discussion

This chapter presents the results of our performance tests and delves into a discussion on application and performance test outcomes. We will dissect the strengths and shortcomings of the current version of ISPE-RCT. The performance evaluation encompasses metrics like CPU and memory usage. Additionally, this chapter will detail the performance metrics of ISPE-RCT when run on a Macbook Pro laptop, followed by a comprehensive discussion of the findings.

6.1 Application Test Discussion

Regarding the examples provided in Chapter 5, ISPE-RCT exhibits certain limitations when implemented in clinical trials. Some of these limitations stem from the inherent logic of the parser, while others arise from the constraints of synchronous languages. Conversely, there are numerous benefits to structuring actions based on SCCharts models. This section delves into the limitations and strengths of ISPE-RCT, shedding light on the underlying reasons.

6.1.1 Advantages

ISPE-RCT is a programming environment for clinical trials. Its syntax and semantics are designed based on English to make learning easy. It is Java-based software, and the code base contains a parser and a web service based on TCP/UDP protocol. Furthermore,

ISPE-RCT comes equipped with pre-configured SCCharts models. The parser is adept at producing executable SCCharts models derived from the ISPE-RCT language code.

Advantages of English-like Syntax

Since ISPE-RCT is designed to resemble natural language, its syntax is intuitive and straightforward. As introduced in Chapter 3, ISPE-RCT has an English-like syntax. The inherent advantage of such a design lies in its accessibility. Leveraging familiar English constructs makes the language more approachable to those without a programming background. It can significantly reduce the learning curve, as users do not have to memorize as many technical terms and symbols. Thus, any clinical trial research team can benefit from programming, regardless of background. It lowers the entry barrier to programming and allows more research team members to participate in the trial design and management process.

Due to the time limitation, this study didn't involve a completed research test to see if participants with different backgrounds could learn and use ISPE-RCT quickly and correctly. However, the future Appendix 2 is how to measure the advantages of English-like syntax.

6.1.2 Advantages of Web Service

ISPE-RCT provides a comprehensive digital environment for designing, conducting, and managing clinical trials. One of its core features, web services, promotes real-time interactions and contactless execution, offering several advantages in the clinical research landscape. This section highlights these benefits in the current global health scenario and the increasing need for digital and remote trial management solutions.

- **Real-time interactions:** ISPE-RCT harnesses the power of web services and network protocols to provide a dynamic communication interface linking researchers, healthcare providers, and trial participants. This functionality promotes the immediate exchange of information, thereby improving the accuracy of data collection and the overall operational efficiency of the trial. Moreover, the following section has a deeper explanation of this real-time feature, as it is supported by SCCharts models.

- **Automated Reminders:** ISPE-RCT sends timely reminders to healthcare providers to administer required interventions using automated processes. This automation continues until the action is confirmed, ensuring critical interventions are not missed due to oversight or human error.

- **Live Monitoring:** ISPE-RCT allows researchers to monitor the real-time status of participants and their data. This capability allows for the early identification of

potential anomalies or problems and, thus, for rapid corrective action. For example, the clinical team needs to monitor every participant's status in the two COVID-19 area clinical trials [103], [104]. Once an adverse event occurs, the clinician must ensure the participants are treated as soon as possible. Live monitoring could lead to improvements in trial outcomes and participant safety.

- **Improved participant engagement:** An interactive environment could potentially increase participant engagement by involving them more actively in the trial process. This active participation could improve their understanding of the trial protocol and increase their adherence to it, contributing to enhanced trial outcomes. For example, most depression area clinical trials must collect questionnaires as the outcome measure.

Advantages of SCCharts Models

As mentioned in the previous subsection, real-time interaction is one of ISPE-RCT's core features. The implementation of real-time interaction involves web service and synchronous programming language. Regarding the synchronous programming language, events are perceived as occurring instantaneously and simultaneously, simplifying concurrent tasks' management. Figure 6.1 is the SCCharts model for doing the same action to 100 participants. For example, when the designer created an action by the following code:

```
1    after 1 day giveDose to group1 with 2 mg
```

The parser of ISPE-RCT will create this action based on this SCCharts model. In this case, for example, the parser will retrieve the group members from Group 1, and we assume there are Participants 1 to 100 in Group 1. This model is equivalent to a combination of 100 single-participant actions. At the initial state, the clock starts counting. After 1440 minutes, this model will remind nurses that participants with labels 1 to 100 need an intervention. Once a nurse finishes dosing participants, they can send a message to the server of ISPE-RCT to tell them the dosing has been done. Then, the web service will read and analyze the content and set the dosing status of Participant 1 to done. Once the clinical team finishes all the participants' dosages, this model will finally move to the done state.

Applying SCCharts models in ISPE-RCT, all events can be processed and finished within a single time unit, or "tick." This immediate processing capability allows ISPE-RCT to respond rapidly to various scenarios and needs, achieving efficient and precise management and operation within clinical trials. Consequently, SCCharts models help ISPE-RCT to handle complex parallel operations in clinical trials simultaneously.

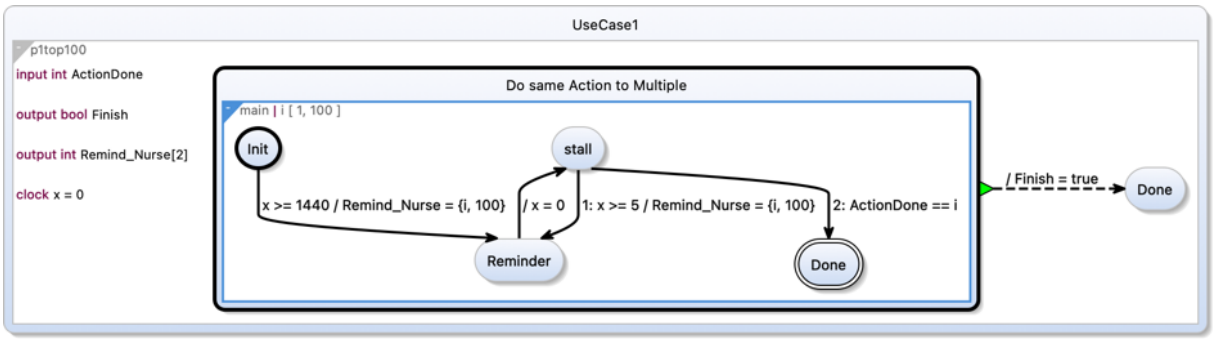


Figure 6.1: SCCharts Model for Doing the Same Action to 100 Participants

6.1.3 Limitations

For instance, a clinical trial might necessitate the simultaneous execution of multitasks across multiple patients. The most prominent disadvantage of ISPE-RCT is that users lack sufficient flexibility when designing their programs. Several cases in the above clinical trials cannot match proper functionality. It is necessary to expand the set of models in SCCharts to increase the versatility and applicability of ISPE-RCT in different clinical trial scenarios. Each model in SCCharts represents a unique event or sequence of events in the clinical trial process. Developing and incorporating more SCCharts models can cover a broader range of clinical trial scenarios, actions, and events.

- **Multitasks across different participants:** ISPE-RCT has an SCCharts model for doing the same operation with different participants. However, for instance, in the parallel study of Fluticasone, the research team wants to collect seven aspects of PROMIS-29 from participants simultaneously. Currently, if the user wanted to collect data on the seven different aspects of PROMIS-29, they would have to write separate lines of code for each aspect. It can become cumbersome and inefficient, especially in larger and more complex trials requiring simultaneous data collection on multiple variables.

  As a synchronous programming language, SCCharts can handle multitasks. The main difficulty is how to design the parser function. Section 7.2 has a more comprehensive discussion.

- **Support for contactless trials:** The need for contactless clinical trials has increased due to global health crises such as the COVID-19 pandemic. ISPE-RCT responds to this need by facilitating remote interactions and digital data collection, thereby minimizing the need for physical contact and the subsequent risk of infection transmission. For example, two COVID-19 area clinical trials above are designed to be contactless trials. The research team, however, needs to ensure the participants get timely intervention. Thus, ISPE-RCT can perform better if it supports another

reminder system sent directly to the participant. In this case, the clinical trial designer can choose to let the participants or clinician team do the intervention.

- **More flexible time arrangement:** ISPE-RCT might not effectively accommodate such complex time-bound actions in the present version. Regarding the time interval arrangement, ISPE-RCT has three models. Firstly, the user can use the immediate execution model by directly designing the action without any time constraints. ISPE-RCT will execute it as soon as the conditions for executing a task are met. Secondly, users can use the keyword 'after' to design a delayed execution, allowing a task to be scheduled for a future time. At last, the user can use the keyword 'every' to design a periodic execution, which is suitable for actions that need to be executed repeatedly at regular intervals.

  However, In the iTBS for Adolescent Depression [99] clinical trial, the research team needed to collect the C-SSRS questionnaire on the baseline, weeks 1, 2, 3, 4, 5,8, and 16. Moreover, the clinical team gives intervention to each participant from Monday to Friday.

- **Support for devices:** One limitation of ISPE-RCT is its need for direct device integration capabilities. It does not natively support direct connections to specific medical or monitoring devices such as iTBS equipment [99] or sleep monitors. The benefits of such integrations would be many. They could enable the automated collection of data from these devices, reducing the need for manual data entry and, thus, the possibility of human error. Similarly, they could enable the system to respond in real-time to changes in patient status detected by these devices, enabling more reactive and personalized study management. They also allow direct management of device settings and operations through the ISPE-RCT language, leading to more consistent and reproducible interventions across trials and sites.

6.1.4 Summary

The study of ISPE-RCT has proved to be a unique endeavour in applying programming concepts to clinical research. Intending to answer two key research questions - the feasibility of designing a programming environment for clinical trials and the applicability of synchronous programming language models within this environment - we have analyzed ISPE-RCT's strengths and areas for improvement.

ISPE-RCT's syntax and semantics, modelled on the English language, facilitate ease of use and make learning accessible to people from diverse backgrounds. Its English-like syntax reduces the learning curve, allowing more research team members to participate in the trial design and management process. In addition, it uses web services for real-time interactions, automated reminders, live monitoring, and improved participant engagement,

providing a comprehensive digital environment for the design, conduct, and management of clinical trials.

Applying SCCharts models within ISPE-RCT supports real-time interactions and rapid response to different scenarios and needs. It also enables efficient and accurate clinical trial management of complex parallel operations.

However, specific improvements could enrich the functionality of ISPE-RCT. The existing setup needs more flexibility in program design, including multi-tasking capabilities between participants and support for contactless trials, which is particularly relevant in global health crises such as the COVID-19 pandemic. Accommodating more complex time-bound interventions and a more comprehensive range of clinical trial scenarios will also be beneficial.

ISPE-RCT also lacks direct device integration capabilities, and future versions could consider incorporating standardized healthcare interoperability protocols to facilitate secure information exchange with a wide range of medical devices and systems. While this presents challenges regarding data security, patient privacy, and the technical complexity of device integration, the potential benefits - automated data collection, real-time response to changes in patient status, and consistent, reproducible interventions - justify these enhancements.

In conclusion, our analysis confirms the feasibility of designing a programming environment for clinical trials, i.e., ISPE-RCT, while highlighting the successful application of synchronous programming language models. Furthermore, it provides valuable insights for further improvement of ISPE-RCT, making it an even more powerful tool for conducting and managing clinical trials.

6.2 Results of Performance Tests and Discussion

In Chapter 5, we discussed the utility of employing OSHI a tool to monitor CPU and memory usage during our tests.

6.2.1 CPU Usage Overview

This section delves into the specifics of the CPU usage recorded during each of our tests and provides a comprehensive discussion of the results. It is imperative to note that all tests were conducted under controlled conditions to achieve the most reliable and accurate results. This entailed closing all non-essential software applications, halting unnecessary processes, and ensuring no superfluous threads ran in the background. The ensuing results thus present a clear picture of the CPU usage specifically attributed to the tasks under examination, uninfluenced by external factors or unrelated computational tasks. The following is a CPU usage overview over 5 minutes.

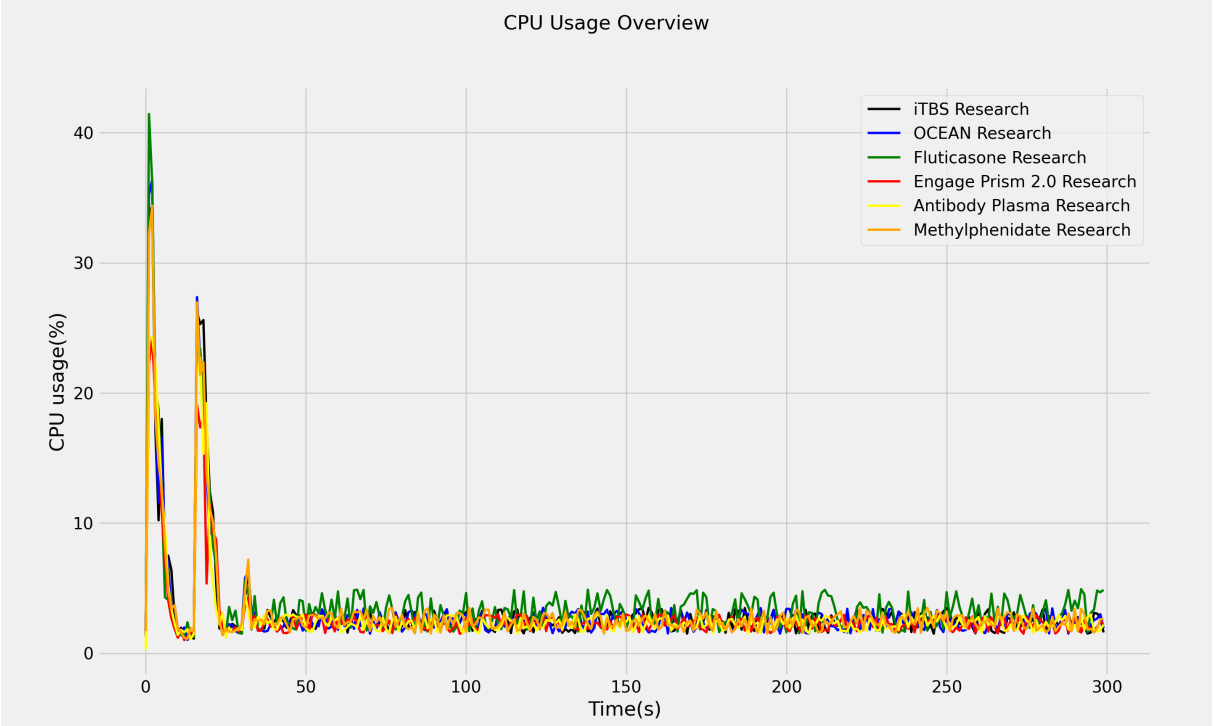


Figure 6.2: CPU Usage Overview

Figure 6.2 presents the CPU usage during the execution of ISPE-RCT, which is constructed based on the programs detailed in Chapter 5. The X-axis denotes time in seconds, while the Y-axis illustrates the percentage of CPU consumption by the Java application. Upon launching the server, a sharp uptick in CPU usage occurs within the initial second. This spike is attributed to ISPE-RCT's generation of SCCharts models, which concludes around the 2-second mark. Subsequent fluctuations in CPU usage, observed between the 3 to 10-second interval, arise due to ISPE-RCT establishing a connection with MySQL and initializing the server on port 3306. With no notable operations between 10 and 15 seconds, CPU usage remains relatively consistent, hovering around 2% to 5% respectively.

A noticeable rise in CPU usage at the 15-second mark corresponds with a client's attempt to connect to the server. The server's task during this phase is to validate the client's credentials against the database before granting access. This verification process lasts roughly 5 seconds, after which system operations stabilize. Another discernible increase, between 30 to 32 seconds, is due to the user transmitting a message to the server, causing CPU usage to surge above 5% momentarily before settling back to around

Regarding figure 6.2, ISPE-RCT's CPU consumption is impressively modest in its idle state, averaging between 2% to 5% and the peak CPU usage ranges between 25% to 45%. This peak arises during the server preparation phase. Moreover, the more SCCharts we create, the more CPU will be used. Therefore, the 6th test has the highest peak of CPU usage. Given that this value doesn't exceed 50%, it is deemed acceptable for software applications [121]–[123].

The ISPE-RCT designer requires a device to run and host the server, enabling other users to connect and communicate. Therefore, we examined the CPU usage during client connections and message transmissions. While isolating the CPU usage changes attributed solely to a new user connecting to the server is challenging, we can assert that the value will not surpass that of server initialization. The CPU usage spike at the 15-second mark encompasses client UI setup, data retrieval from the MySQL server, and the client-server connection process. Yet even under these conditions, the CPU usage doesn't exceed 30% when connecting the client to the server. Most importantly, sending a message—arguably the most utilized feature—only demands less than 5% CPU usage.

In summary, ISPE-RCT doesn't heavily tax the CPU when hosting the server and overseeing the clinical trial. The CPU consumption for all operations remains within an acceptable range. Interestingly, while the Fluticasone research doesn't boast the most extensive collection of SCCharts models, it does register the highest CPU usage. Upon closer examination, we observed that this research involved a substantial number of participants, totalling 1407. This led us to hypothesize that the peak CPU usage depicted in the graph might be attributed to the initialization of these participants. Consequently, we delved deeper into analysis in section 6.2.3.

6.2.2 Memory Usage Overview

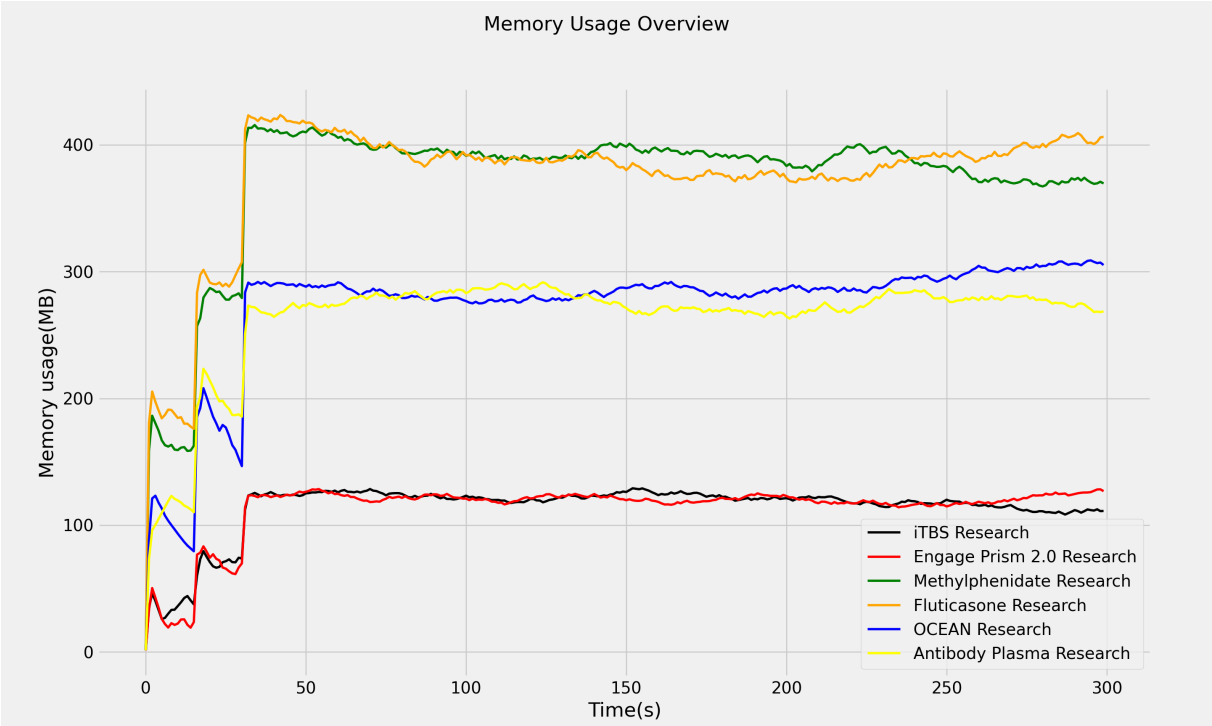


Figure 6.3: Memory Usage Overview

The memory usage tests inherit the criteria of CPU usage tests. To ensure the accuracy and reliability of the results, these memory usage tests were performed in isolation, with no other software in operation.

Figure 6.3 is the memory usage of single group assignments. We did the memory and CPU usage tests simultaneously, so the three actions were performed in the same seconds as above. As in the figure, we can find three moments that have a more rapid increase.

We started the server initially. Memory usage increased rapidly in the first two seconds of both tests, with more than 100 MB for test 1 and around 75 MB for test 2. Since the first test has more SCCharts models, the server needs more memory to generate, load, and execute these models. At time 15 seconds, a new user connects to the client side, and these two figures have a similar significant increase. The clinical trial designers run the program and server in a real clinical trial, and they will not join as users of the client UI. We can consider it as the increase, including the loading of the client UI, the connection from the client UI to the server, and the loading of client data at the server. That's why we got a more significant memory usage increase at 15 seconds than at the initial time. The memory usage increase of sending messages in the two tests is similar.

After every action is completed, memory usage will slightly decrease. There are several reasons for the decrease. One is Java has garbage collection mechanisms [124], [125]. The garbage collector automatically reclaims its memory when an object is no longer referenced. Another reason can be the termination of sub-processes or threads. Since ISPE-RCT has multiple processes and threads, the memory associated with any of those processes or threads will typically be released once they're terminated.

Regarding the curves in the figure, we can see some fluctuations in memory usage after a decline. As we tested, the memory will remain stable if there are no more connections and communications. Nevertheless, as an interactive environment, this case will not frequently occur during a clinical trial. Thus, we will not discuss the stable level.

Among the tests, iTBS research and Engage Prism 2.0 research consumed the least amount of memory, while Methylphenidate research and Fluticasone research demanded the most. The primary driver behind this discrepancy is the number of participants and complexity of SCCharts models created by ISPE-RCT. Specifically, Methylphenidate research incorporated a particularly intricate SCCharts model, necessitating waiting for a scale result to decide upon dosage alterations. This added complexity naturally led to a higher memory footprint.

Throughout the six memory tests, the memory consumption observed at the 3-second mark ranged between 50 MB and 210 MB. Meanwhile, the total memory usage fluctuated between 120 MB and 410 MB. When assessing the memory demands of connecting the client UI to the server, ISPE-RCT required approximately 60 to 100 MB. Across all six tests, a consistent 50 MB of memory was used for sending and receiving messages. No-

tably, the memory requirements for server setup varied considerably, primarily influenced by the quantity and complexity of SCCharts models.

Given these findings, it is essential to contextualize them against industry benchmarks. Memory usage can impact a software application's performance and responsiveness. Software applications, especially those running on server environments, are considered efficient if they maintain their memory footprint within a specific range, which ISPE-RCT appears to do. Our observed numbers align well with acceptable memory usage thresholds for server-based applications [121]–[123], [126], [127], emphasizing the efficiency of ISPE-RCT.

6.2.3 CPU Usage Further Tests

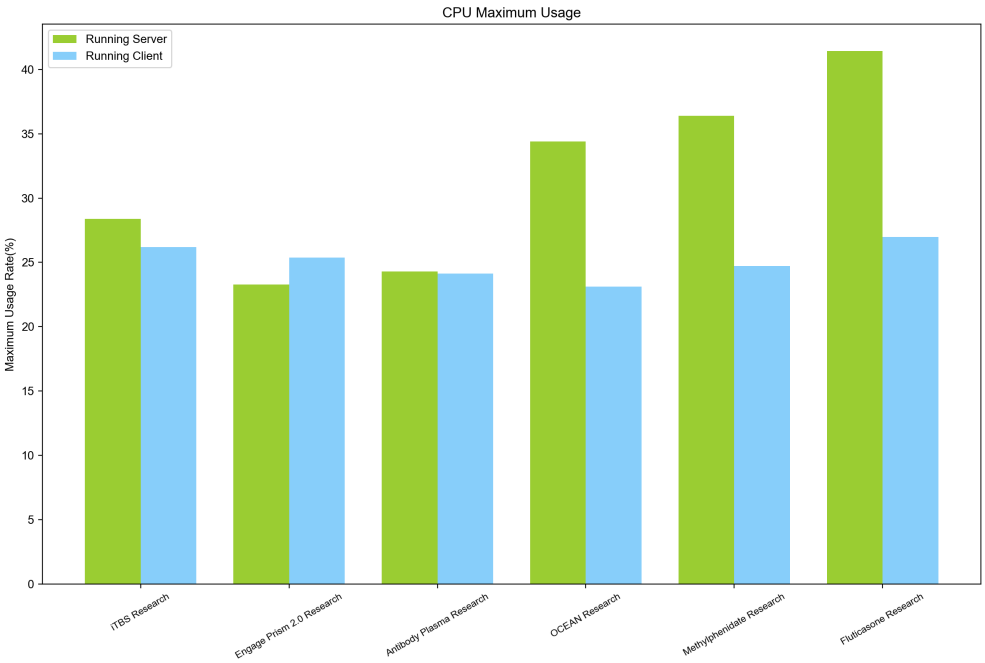


Figure 6.4: Comparison of Maximum CPU Usage

Figure 6.4 offers a comparative analysis of the peak CPU usage during the server or client's operation. The X-axis labels the research studies, while the Y-axis quantifies the percentage of CPU utilization by the Java application. The green bars signify the CPU usage when we run the server, and the blue bars correspond to the CPU consumption during the execution of a client UI.

Moving from left to right, the X-axis shows an increasing number of research participants: 5, 8, 56, 108, 181, and 1407. Correspondingly, the number of their SCCharts models are 7, 4, 4, 6, 11, and 4, respectively.

For participants, the graph indicates that when running the client, the CPU usage rate consistently hovers around 25%. Given that this represents the peak instantaneous CPU usage, it is a commendable performance for most participants.

From a designer's perspective, observing the green bars, it is evident that the CPU usage rate for Fluticasone Research is higher than the others. Based on our prior discussions, we know that the Fluticasone Research involved 1407 participants, far surpassing that of other clinical trials. This explains why its usage rate exceeds that of the Methylphenidate Research, which has 11 SCCharts models. We discerned that the factors influencing designers' server-side CPU usage rate are the number of participants and the SCCharts. We aim to test these two factors separately concerning their impact on server utilization. Since SCCharts models are generated at the outset, our testing for this aspect primarily focuses on the peak CPU usage when the server is initially launched. However, for participants, their most frequent operation is sending messages. Therefore, we intend to examine how the server's CPU usage fluctuates as more participants send messages simultaneously.

CPU Usage among Different Levels of Participants and SCCharts Models

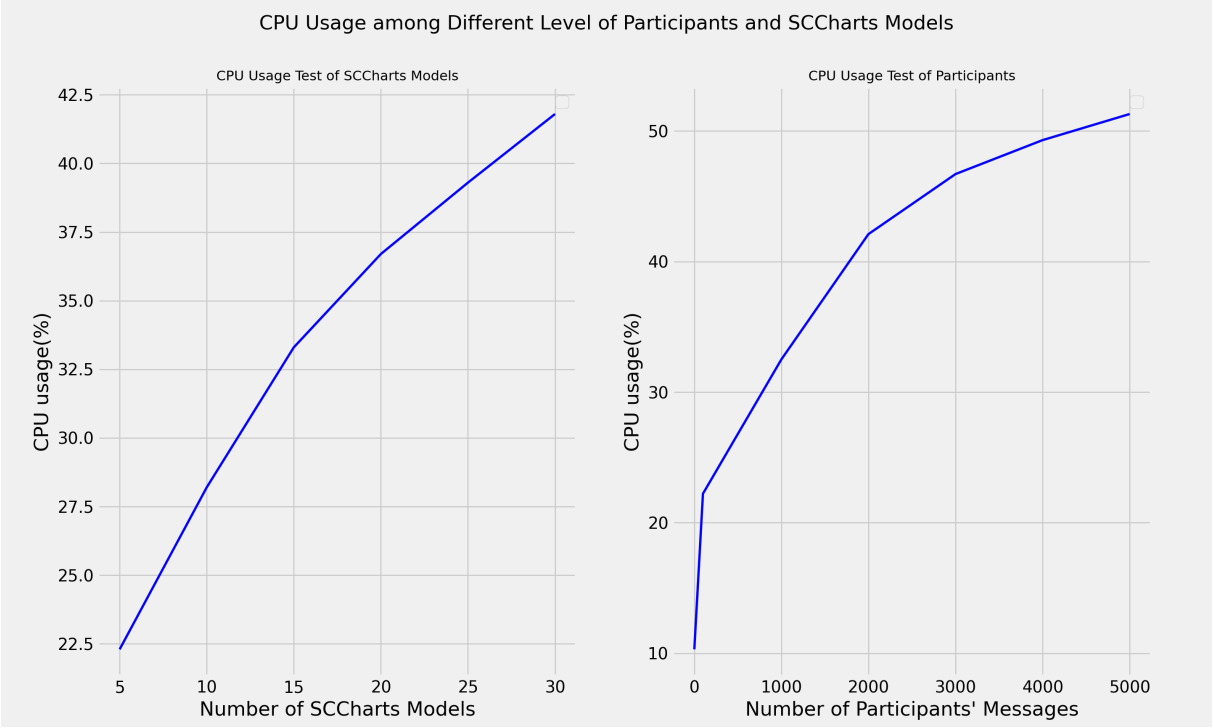


Figure 6.5: CPU Usage among Different Levels of Participants and SCCharts Models

Figure 6.5 depicts the CPU consumption corresponding to varying numbers of participants and SCCharts models.

The graph on the left illustrates the impact of varying numbers of SCCharts models on CPU Usage. To minimize the influence of participant numbers on this test, all data

was derived based on 100 participants. The x-axis represents the quantity of SCCharts models, while the y-axis denotes the peak memory usage when running the server. Their relationship closely mirrors a linear correlation. This is attributed to the fact that the entire ISPE-RCT framework is built upon SCCharts models. Hence, adding each SCCharts model incrementally elevates the CPU usage by the parser, model generation, and server-side model execution. However, it is noticeable that the growth rate decelerates as the number of models increases. This deviation from a perfect linear relationship is likely due to inherent optimizations in Java and SCCharts. Optimizing the parser and web service can mitigate the linear growth of CPU usage with an increasing number of models. This is one of our future directions in section 7.2 Moreover, given that the SCCharts models used in this experiment serve 100 participants simultaneously, adding a model in another programming language would be equivalent to concurrently adding 100 programs in that language, exerting significant strain on the CPU. This underscores the efficiency of SCCharts in handling synchronous tasks. Furthermore, employing a programming language to design clinical trials enhances efficiency, as a single model can cater to all participants. Typically, clinical trials will not have an excessive number of SCCharts models. For instance, the maximum model count was only 11 among the six previously discussed tests. Thus, ISPE-RCT is not constrained by computational power.

The graph on the right delineates the influence of different participant numbers on CPU Usage. The x-axis signifies varying numbers of participants sending messages to the server simultaneously, while the y-axis represents the corresponding peak memory usage. Similarly, to minimize the influence of SCCharts models on the data, all data in the right graph was based on two SCCharts models.

It is evident that with few participants, CPU usage sees a significant surge when we add a new participant. However, as the number of participants escalates, the growth rate of CPU usage noticeably diminishes. Part of this trend can be credited to Java's inherent optimizations. More crucially, the synchronous programming language and the optimization of SCCharts models when processing multiple messages concurrently play pivotal roles.

A large clinical trial may involve thousands of participants. Each participant may receive around 5 messages per day, depending on the trial, and ISPE-RCT also needs to process thousands of input messages from the clinical team. Thus, we tested a maximum of 5,000 messages simultaneously. The result of 5,000 messages is 51.8%, which is still acceptable for software. This underscores the enhancement that synchronous programming language brings to ISPE-RCT. It substantiates our claim that adopting SCCharts in a clinical trial programming environment is a viable strategy for seamless scalability.

6.2.4 Memory Usage Further tests

Given the memory test presented in Figure 6.3, there isn't a pronounced reduction in memory usage after the completion of each action. Consequently, we aim to assess the memory increment during each action until the first decline in memory usage is observed. Figure 6.6 shows the result of Memory usage.

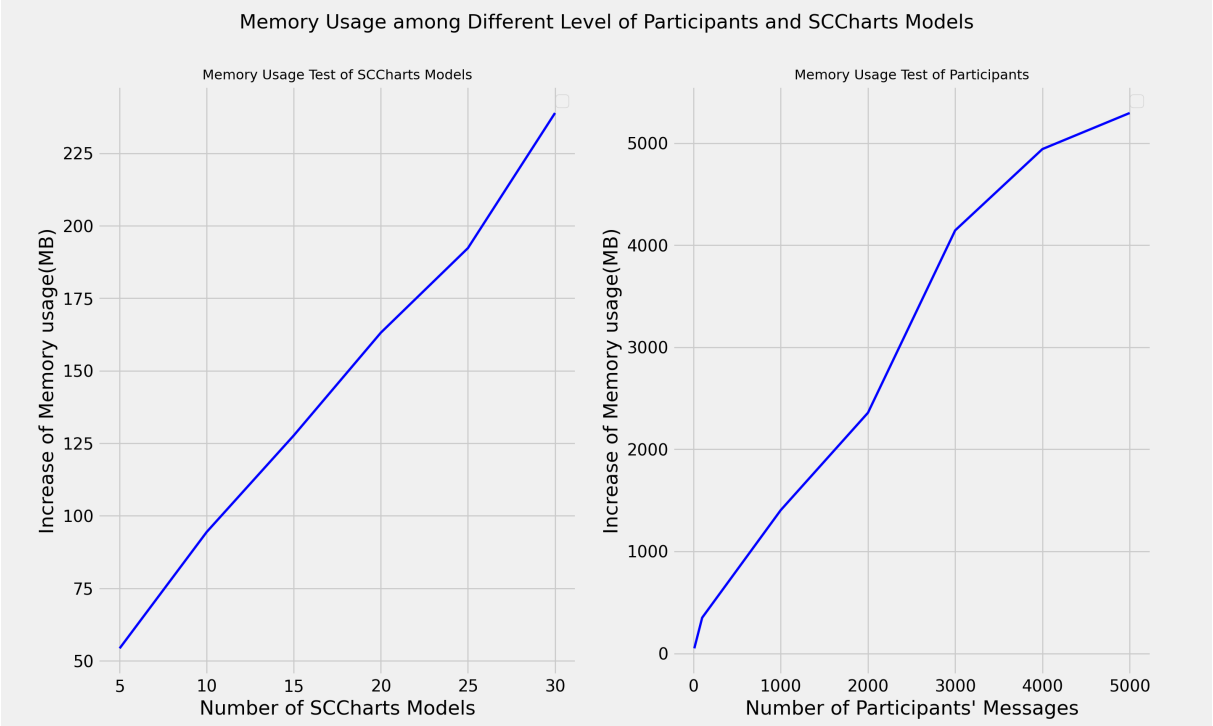


Figure 6.6: Memory Usage among Different Level of Participants and SCCharts Models

The data on Memory Usage largely aligns with the conclusions drawn from CPU Usage. However, extensive data reading/writing and network communications, among other IO operations, may necessitate temporary data storage, leading to increased memory consumption. The left graph illustrates the change in memory usage when running the ISPE-RCT server. In contrast, the right graph depicts the variation in memory usage as different numbers of participants send messages simultaneously. The x-axis for both graphs mirrors CPU Usage, but the y-axis now represents the incremental change in memory usage.

The results in the left graph are based on data from 10 participants. A near-linear growth is evident, stemming from the fact that as the number of SCCharts models increases, the corresponding memory requirement also exhibits a linear rise. This is consistent with the findings from the CPU Usage analysis. The deviations from a perfect linear trend can be attributed to Java and the parser optimisations.

The right graph is derived from data under the premise of 5 SCCharts models. When processing multiple messages simultaneously, a clear linear relationship isn't apparent.

However, it is discernible that as the number of messages to be processed simultaneously increases, the incremental memory usage diminishes. ISPE-RCT first logs the message and then proceeds with its processing and analysis. While this is influenced to some extent by SPL optimization, it is predominantly a result of inherent Java optimizations.

Furthermore, since ISPE-RCT features offline message logging, messages that do not alter the SCCharts models' signals must be written into MySQL. This can consume a significant amount of memory. For instance, when handling 5000 messages simultaneously, the memory increment might exceed 5000MB. However, this is just a transient spike. Due to Java's inherent mechanisms, the system will subsequently release unnecessary memory. Thus, if a clinical trial designer aims to run a trial involving thousands of participants, they would require a computer with at least 16GB of memory.

6.2.5 Summary

Several pertinent findings emerged in our exhaustive evaluation of the ISPE-RCT's CPU and Memory usage performance across single-group and parallel assignments. These observations were attained under strictly controlled test conditions to ensure accuracy.

ISPE-RCT's CPU usage exhibited an initial spike during server initiation, attributed primarily to the generation of SCCharts models. While notable, this spike consistently remained below 50% of the available CPU resources, an acceptable threshold per industry benchmarks. Beyond the initialization phase, operations such as the connection to MySQL, server configuration, and client-server interactions only introduced modest fluctuations in CPU usage. The most intense operations, like client validation, only saw 30% CPU consumption peaks. Parallel assignment outcomes were largely congruent with single-group results, indicating the consistent behaviour of ISPE-RCT across various testing paradigms. Across scenarios, the application displayed impressive efficiency, particularly with CPU consumption hovering between 1% and 3% during idle states.

Memory usage exhibited three distinct moments of rapid increase: during server startup, client connection, and message transmission. However, after each activity concluded, a noticeable drop was observed due to Java's inherent garbage collection mechanism and termination of specific subprocesses or threads. When analyzing memory usage during parallel assignments, the most significant consumption was recorded during tests with intricate SCCharts models, and Test 6 displayed the highest memory footprint. A comprehensive review of all tests revealed a memory consumption ranging from 50 MB to 210 MB at the 3-second mark and an overall usage between 120 MB and 410 MB. Notably, around 50 MB was consistently used across tests for message transmissions.

Adopting SCCharts models and synchronous programming offers distinctive advantages, particularly for multi-task and parallel-task operations. SCCharts facilitate clearer representation and manageability of complex tasks, promoting deterministic behaviour.

Synchronous programming ensures tasks run in a lock-step manner, fostering predictability and reducing the potential for race conditions or deadlocks. The synchronous execution means CPU and memory usage do not experience drastic fluctuations, even when multiple tasks run in tandem. This predictable resource consumption, coupled with the structured nature of SCCharts, ensures that ISPE-RCT remains efficient even during intensive synchronous operations.

The detailed assessments of CPU and memory consumption highlight the strengths of SPL in enhancing efficiency and minimizing excessive CPU and memory utilization.

ISPE-RCT showcases efficiency and optimization. Incorporating SCCharts models and synchronous programming enhances clarity and manageability and optimizes resource consumption, making ISPE-RCT an ideal tool for randomized controlled clinical trials.

7 Conclusion

This thesis dives into the design of a programming environment for RCTs, ISPE-RCT, which has three main features. With English-like syntax and semantics, ISPE-RCT constitutes an accessible tool for clinical teams without coding experience to program for their clinical trials. ISPE-RCT also features built-in SCCharts models developed based on RCT requirements and a parser to distinguish the keywords from English-like language and parse each action to a proper SCCharts model using its internal functions. The third feature is the web service, the core module enabling interactions. The program with SCCharts models runs on the server from initiating clinical trials until the end. With the server active, users can send messages or files to each other and the server. The server reads messages from clinical team users to update the dosage status and collect participant data. Having presented the whole design of ISPE-RCT, this final chapter points out future research directions, including possible ISPE-RCT improvements, un-performed research tests, and relevant measurements.

7.1 Achievements

The major achievements of this study are highlighted as follows:

- **English-like syntax for the programming environment:** Based on existing research on clinical trial requirements and the involved programming languages, we created an English-like programming language to meet clinical trial team needs.

- **Applied synchronous programming language models:** To meet the needs of parallel RCT assignments, we successfully introduced SCCharts models into the designed programming environment to make it a more secure and efficient system.
- **Parser:** A parser is included to achieve the transformation from the English-like programming language to executable SCCharts models.
- **An interactive programming environment for RCTs:** We designed a web service for user interactions.
- **Research tests on completed clinical trials:** We applied ISPE-RCT to six levels of completed clinical trials to verify its limitations and strengths.
- **Research tests on the CPU and memory usage of ISPE-RCT:** We monitored the CPU and memory usage of ISPE-RCT using OSHI. Benefiting from SCCharts models, ISPE-RCT has reasonable CPU and memory usage, especially when applied to parallel tasks.

Thus, we have designed a new programming environment for RCTs and successfully incorporated the synchronous programming language models.

7.2 Future Directions

7.2.1 Limitations and Solutions

According to the analyses and discussions in Chapter 6.1, ISPE-RCT has certain limitations applied to the completed clinical trials, which are discussed here, and solutions to the restrictions are provided.

- **Mulititasks across different participants:** As mentioned in section 6.1.3, we intend to design a multi-task action with only one line of code and enable ISPE-RCT to execute similar tasks.

  A flexible implementation could involve expanding ISPE-RCT to include loops or vectorized operations, thus allowing users to specify a list of necessary actions and then loop through it and execute each operation. In this way, users can execute any number of actions in any combination.

- **Support for contactless trials:** Diseases such as COVID-19 require contactless clinical trials to guarantee the safety of the research team. Currently, the ISPE-RCT web service supports communication between different users, allowing participants or their guardians to complete the interventions. Therefore, one possible solution is adding a user group table in the MySQL database to divide them into different

groups. Moreover, a suitable method is desired to call the function and set which user group's messages are valid with a Boolean value in the table. The server also needs a function to distinguish which user group is sending messages.

Assuming the designer sets the system to accept messages from the clinical team and the participants, the server will do the following when the SCCharts model waits for an "Action Done" signal to stop the alert looping. The server receives all user messages; The server retrieves user details from the database and reads the column of validation; If the Boolean value equals 1, then the system analyzes the message and jumps to the original message analysis function; Otherwise, the system does nothing. In this case, both participants and the research team can send an "Action Done" message to the server.

In summary, by adding the ability to update user permission and the corresponding server function, ISPE-RCT allows the designer to choose who performs the intervention.

- **More flexible time arrangement:** The internal clock of each SCCharts model can be harnessed to regulate the scheduling and execution of tasks in the clinical trial environment. This internal clock also allows for fine-tuning the events to precise time intervals, enhancing the system's ability to handle tasks requiring immediate execution, delayed execution, or periodic repetition.

  However, setting the current ISPE-RCT to support actions repeated every Monday to Friday is challenging, as the clock "ticks" every second, while all the models are created at the beginning of clinical trials. Users have to design many lines for the Monday to Friday actions. A possible solution is creating a function in the parser to read days from Monday to Sunday and generate several models to achieve this action. In this case, users only need to highlight periodical days in a week to design the actions.

- **Device support:** As discussed in section 6.1.3, SCCharts models are desired to support certain devices. Considering the different features of various devices, implementing these capabilities may require cooperation with device manufacturers to develop specific ISPE-RCT modules or extensions for each device.

  However, it is essential to note that these potential improvements come with their challenges, including data security, patient privacy, and the technical complexities of device integration. These issues must be carefully considered and addressed when designing and implementing the corresponding extensions to the ISPE-RCT language.

7.2.2 Flexible

ISPE-RCT seeks to redefine the landscape of clinical trial design through its unique interactive and programmatic approach. However, its promise depends on its multi-faceted adaptability. Apart from the above-mentioned time management within ISPE-RCT, there are still other areas for improvement.

Task sequencing is another vital area. Clinical trials often have different structures depending on their objectives and designs. ISPE-RCT should provide a dynamic task sequencing feature where researchers can prioritize, re-order, or introduce new tasks on the fly. This function would accommodate everything from simple drug efficacy tests to complex multi-drug, multi-phase trials.

Group allocation tools should cover different randomization methods. The system should also handle the nuances of single-blind, double-blind, and open-label studies, ensuring data integrity and participant blinding.

Settings for variable study periods are also essential. Be it a cross-over study where participants receive multiple treatments sequentially or a factorial study where various treatments are tested simultaneously, ISPE-RCT must be able to adapt the study design seamlessly.

The integration capabilities of ISPE-RCT must be extensive to position it as a holistic tool, meaning pulling data from electronic health records, laboratory systems, and patient monitoring tools. The built-in feedback mechanism should include a bug-reporting tool and a channel for researchers to suggest additional features, modifications, and enhancements based on real-world use.

Finally, the importance of a sophisticated alert system must be considered. Safety is paramount in trials, and ISPE-RCT should be able to set different alert levels from minor deviations in expected outcomes to critical safety violations so as to ensure timely intervention and safeguard participant well-being.

In summary, while ISPE-RCT promises to reshape the landscape of clinical trial design with its interactive and programmatic approach, its success depends on its flexibility and adaptability to the vast and varied domain of clinical research. Ongoing updates, informed by direct user feedback and ongoing research needs, will be essential to ensure its longevity and effectiveness.

7.2.3 Future Research Test Plans

The primary objective of previous tests was to rigorously assess the suitability of ISPE-RCT for a broader spectrum of RCTs. We also ventured to determine its performance standards as a software utility.

A particular focus was given to the user-friendliness of ISPE-RCT, especially for professionals in medicine. By employing an English-like syntax, the aspiration was to facil-

itate an intuitive interface, allowing users, even those without programming knowledge, to swiftly learn and adeptly use the tool.

Empirical observations from the initial tests indicate that ISPE-RCT, enhanced by the synchronous programming language SCCharts, exhibits exemplary CPU and memory performance, standing well within the industry standards. This underscores the software's efficiency in handling single-group and parallel-task RCTs.

However, a comprehensive evaluation of a tool would be incomplete without assessing its ease of usage, particularly for its target user base in medicine. While we intended to delve deeper into this aspect, the user acceptance test remains unfinished due to constraints on time and recruitment, which is an inevitable limitation of this research.

Nevertheless, recognizing the importance of this evaluation item, we have detailed a test plan to assess the user-friendliness and acceptability of ISPE-RCT's English-like syntax. The research test plan is included in Appendix 2 for subsequent research to employ, thus offering a comprehensive understanding of ISPE-RCT's user-centric design and applicability in real-world RCT scenarios.

7.3 Conclusion

This study sought to design a programming environment dedicated to RCTs and to apply SCCharts models to achieve synchronous behaviors in clinical trials. ISPE-RCT provides an easy way for researchers to quickly learn and use, even for those with no programming experience. Combining the advantages of natural and programming languages provides a more concise way for researchers to design clinical trials. The ISPE-RCT design also considers the needs of RCTs, including group randomization, data collection, and bias reduction. ISPE-RCT has also been tested to verify its applicability, identify its areas for improvement, and provide feasible solutions.

A salient feature of the designed system is incorporating web services, which foster an enriched user experience and promote more profound human-computer interactivity. The synergy between SCCharts models and web services brings to the fore an intelligent alert system, ensuring participants are provided with real-time interventions, a cornerstone for the success of any RCT.

The overarching result from both CPU and memory analyses is clear: ISPE-RCT, bolstered by the synchronous programming language SCCharts, offers a reliable, efficient, and scalable environment for RCTs. It capably handles serial and parallel tasks while maintaining modest and acceptable resource consumption. Be it single-group clinical trials or more complex parallel assignments, ISPE-RCT stands out as a well-optimized and dependable tool.

Unfortunately, the user acceptance test of ISPE-RCT was not successfully completed due to time and recruitment constraints. However, we have included the testing plan in Future Direction 7.2.

This study successfully implemented an interactive and synchronous RCT programming environment rooted in an English-like syntax. Incorporating SCCharts into the ISPE-RCT fabric is not just an academic exercise but a performance optimization.

Appendices

8

Introduction of Completed Clinical trials

8.1 iTBS for Adolescent Depression

8.1.1 Introduction

- **Design:** This pilot study focuses on the efficacy, durability, safety, and feasibility of using Intermittent Theta Burst Stimulation (iTBS) in treating adolescent depression. The design includes a systematic investigation spread across 16 weeks, consisting of a pre-treatment screening visit, 20 iTBS treatment sessions over four weeks, and three preplanned follow-up visits up to 12 weeks post-treatment.

- **Evaluation Method:** The evaluation of the treatment's efficacy and durability primarily hinges on observed changes in clinical ratings. These ratings are captured before, during, and post-treatment using several metrics, such as the Children's Depression Rating Scale-Revised (CDRS-R), the Hamilton Depression Rating Scale (HAM-D), and Non-Suicidal Self-Injurious Behavior (NSSIB) measures. An expected outcome is the improvement and persistence of reduced depressive symptoms throughout the follow-up period.

- **Safety Evaluation:** Safety is another crucial aspect of this study. The investigators will measure any changes in suicidality (thoughts and behaviors) through the Columbia Suicide Severity Rating Scale (C-SSRS) and a psychiatrist's clini-

cal assessment. It is hypothesized that iTBS treatment will reduce suicidality and NSSIB.

- **Feasibility Criteria:** Feasibility is assessed through treatment completion rates and patient withdrawal occurrences. A minimum of 75% treatment completion rate by all subjects and no more than one withdrawal due to intolerable side effects or persistent MDD symptoms are set as feasibility criteria.

This integrated approach of simultaneously investigating different aspects is designed to provide comprehensive preliminary data to justify more extensive future studies into iTBS's role in treating adolescent depression.

8.2 The Feasibility Engage Therapy With Video Support for Homebound Older Adults

8.2.1 Introduction

This study used the combination of "Engage" and "Personal Reminder Information and Social Management"(PRISM), called Engage-Prism.

"Engage" is a treatment protocol designed by Alexopoulos et al. [128], [129] for older adults' depression that has been developed to be easily administered by community clinicians. This therapeutic approach is grounded in the concept that late-life depression arises from dysfunctions in the brain's positive valence systems, which process rewarding stimuli. "Engage" employs "reward exposure" as its primary intervention, encouraging patients to participate in social and physical activities that are personally meaningful and rewarding.

PRISM system is a specially designed computer system for older adults to provide support and reduce social isolation. Czaja et al. [130] evaluated the PRISM system in a multi-site randomized field trial involving 300 older adults living independently in the community who were at risk for social isolation. The study shows that the PRISM system reduces feelings of loneliness, increases perceived social support, and enhances well-being.

8.3 COVID-19 Study of Repurposed Medications - Arm C (Fluticasone)

8.3.1 Introduction

This study aims to evaluate the efficacy of repurposed medications in mitigating symptoms in non-hospitalized patients with mild to moderate COVID-19. The study is designed to

be remote, with participants self-reporting any new or worsening symptoms or medical events experienced while taking the study drug or placebo. Physical visits will only occur if necessary for the participant's well-being by the study team.

The study is part of a platform protocol designed to generate evidence for prioritized drugs repurposed from other uses with solid safety records. It shows preliminary signs of clinical efficacy for COVID-19 treatment. The ultimate objective is to determine whether these repurposed medications can improve patients' conditions more rapidly and decrease death and hospitalization rates due to COVID-19.

The study participants, confirmed to have SARS-CoV-2 via PCR or antigen test in an outpatient setting, will be randomly assigned to receive the study drugs or placebo. This allocation will be based on the arms actively recruiting during randomization. Study drugs may be added or removed based on an adaptive design and emerging evidence.

The study is double-blinded, with participants randomized to either the study drug arm or placebo arm along with the standard of care. All participants will receive a complete supply of the study drug or placebo, depending on the group to which they've been assigned. The randomization process will be manipulated to maximize placebo data across arms, and data will be analyzed via shared placebo.

Though designed to be remote, screening and enrollment may occur in person at specific sites, and unplanned study visits may appear in person or remotely as needed for safety reasons. Participants will be asked to complete questionnaires and report safety events during the study. An online system will prompt them to report any safety incidents for review and confirmation via medical records and site staff.

8.3.2 How is the study designed?

This study has an experimental group and a placebo comparator group. Table 8.1 shows these two arms and their treatment.

Interventional Model Description: Double-Blind, Placebo-Controlled

Participant Group/Arm	Intervention/Treatment
Experimental: Arm C - Fluticasone Fluticasone is a self-administered inhaled	Drug: Fluticasone Fluticasone furoate is an inhalable powdered medication comprised of the synthetic medication comprised of the synthetic tri-fluorinated corticosteroid, fluticasone furoate. This compound, insoluble in water, appears as a white

Continued on next page

Table 8.1 – continued from previous page

Participant Group/Arm	Intervention/Treatment
drug. Participants will self-administer 200 µg (1 blister) of fluticasone once daily for 14 days. The powder in the blister is revealed once the inhaler is activated and the subject inhales the study medicine through the mouthpiece	powder. It is supplied in a two-tone grey inhaler equipped with a mouthpiece cover and distinct foil blister strips. The inhaler is stored in a moisture-resistant foil tray that includes a desiccant and a peel-off lid. All packaging is marked to signify that the contents are for experimental use. Participants will self-administer 200 µg (equivalent to 1 blister) of fluticasone furoate daily for 14 days. Alternate names: Fluticasone Furoate
Placebo Comparator: Arm C - Placebo Placebo is a self-administered inhaled agent. Participants will self-administer 1 blister of placebo once daily for 14 days. The powder in the blister is revealed once the inhaler is activated, and the subject inhales the placebo through the mouthpiece.	Other: Placebo Every study arm will incorporate a placebo for comparison. This placebo will closely resemble the study drug in appearance and will be administered in the same manner and dosage. Nevertheless, the placebo will be a non-active substance and will not contain any elements of the drug.

Table 8.1: Arms and Interventions of Fluticasone

8.4 Antibody Plasma Research Study in Hospitalized Patients

8.4.1 Introduction

The COVID-19 Antibody Plasma Research Study in Hospitalised Patients (UNC CCP RCT) was a landmark research project primarily designed to evaluate the safety and

efficacy of Convalescent COVID-19 Plasma (CCP) as a treatment option for individuals hospitalized with confirmed COVID-19 infection. The study started on 27 August 2020 and ended on 4 June 2021, with 56 participants enrolled over the course of the study.

This interventional phase 2 clinical trial followed a double-blind, randomized protocol to ensure unbiased and reliable results. The basic idea was to test whether administration of CCP, which contains antibodies against the SARS-CoV-2 virus, could help a patient's body fight the infection more effectively.

Participants in the trial were treated with two units of CCP, transfused intravenously one at a time, 4 to 24 hours apart. Importantly, the CCP was derived from people who had recovered from COVID-19 infection and therefore had antibodies and possibly other viral inhibitory properties. An interesting aspect of the study was that participants were randomly assigned to receive either two units of CCP with standard antibody levels, as suggested by the FDA, or two units with higher than standard antibody levels. The aim was to investigate whether the antibody level in the CCP played a role in treatment outcome.

The study followed a strict timeline. Patients were randomized within 48 hours of admission to a COVID-19-dedicated service and received convalescent plasma within 24 hours of randomization. Researchers conducted safety and efficacy assessments before, during, and after transfusion and collected samples for further research until day 28 of the study or until the patient was discharged. After discharge, participants could provide longitudinal samples at 1, 3, and 6 months post-transfusion, contributing to a comprehensive data set for analysis.

Given that this study took place in the midst of a global pandemic, its results are significant. The overarching hope is that the knowledge gained from this research will be crucial in informing future strategies for treating COVID-19, thereby alleviating the enormous burden placed on healthcare systems worldwide by the ongoing pandemic.

8.5 Omega 3 for Treatment of Depression in Patients With Heart Failure (OCEAN)

8.5.1 Introduction

Omega 3 for Treatment of Depression in Patients With Heart Failure (OCEAN) is an ambitious clinical trial designed to investigate the potential therapeutic effects of Omega-3 supplements on depressive symptoms in patients with heart failure. Designed specifically for patients diagnosed with moderate to severe Major Depressive Disorder (MDD), the trial aims to provide new insights and perhaps a new treatment paradigm into the complex interplay between heart failure and depression.

However, as with all ground-breaking research, the OCEAN study is a complex, multi-faceted investigation. Several clinical trials and academic papers have been produced under its aegis, each of which sheds light on different aspects of this research question. Some studies focus on the clinical efficacy of Omega-3 supplementation, while others examine its biochemical effects, including its impact on red blood cell omega-3 levels.

This study is part of the OCEAN study. Its central hypothesis is twofold: first, that Omega-3 supplements would be more effective than a placebo in alleviating depressive symptoms, suggesting a potential avenue for incorporating these supplements into the treatment plan for heart failure patients suffering from MDD. Secondly, the study will examine the effects of the different components of Omega-3 supplements. Specifically, it will look at whether pure eicosapentaenoic acid (EPA), one of the key components of omega-3, would be better than a mixture of EPA and docosahexaenoic acid (DHA) at reducing depression.

In addition, the trial addresses the important safety issue by looking at the incidence of adverse events in patients taking Omega-3 supplements. It also provides a comparative analysis of n-3 polyunsaturated fatty acids (n-3PUFAs) and traditional antidepressants, an important factor in assessing the practical feasibility of introducing n-3PUFAs as a mainstream treatment option.

As the OCEAN study progresses, its findings will undoubtedly add considerable value to our understanding of heart failure, depression, and the role of omega-3 supplements in treating these conditions. As the global healthcare community eagerly awaits further developments from this research, the initial findings suggest cautious optimism about the potential of Omega-3 supplements in treating depression in heart failure patients.

8.5.2 How is the study designed?

Masking: Quadruple (Participant, Care Provider, Investigator, Outcomes Assessor)

Participant Group/Arm	**Intervention/Treatment**
Active Comparator: 2:1 EPA/DHA	Drug: 2:1 EPA/DHA 400 Eicosapentaenoic acid/200 docosahexaenoic acid fish oil at 2 grams daily for 12 weeks for participants with chronic heart failure(CHF) and major depressive disorder. Other Names: 400 EPA/200 DHA 2 grams

Continued on next page

Table 8.2 – continued from previous page

Participant Group/Arm	Intervention/Treatment
Active Comparator: High EPA	Drug: High EPA Almost pure EPA at 2 grams daily for 12 weeks for participants with chronic heart failure(CHF) and major depressive disorder. Other Names: An almost pure Eicosapentaenoic acid 2 grams
Placebo Comparator: Placebo	Other: Placebo Matched placebo corn oil capsules at 2 grams daily for 12 weeks for participants with chronic heart failure(CHF) and major depressive disorder. Other Names: Matched placebo corn oil capsules

Table 8.2: Arms and Interventions of EPA/DHA

8.6 Effectiveness of Methylphenidate in Improving Cognition and Function in Older Adults With Depression

8.6.1 Introduction

The Effectiveness of Methylphenidate in Improving Cognition and Function in Older Adults With Depression is an innovative clinical trial aiming to address the continuing challenge of treating residual depressive symptoms in older patients. Recognizing that less than half of older adults with depression achieve a complete remission or functional recovery with first-line antidepressant treatment, this study investigates the potential benefits of combining Methylphenidate (MPH) with the conventional antidepressant citalopram.

The central premise of this trial is to examine the efficacy and safety of Methylphenidate, a drug commonly used to provide immediate relief of depression, fatigue, and apathy in the elderly and medically frail. The trial aims to determine whether MPH, when combined with citalopram, can improve cognitive function and speed recovery, ultimately leading to better clinical outcomes than when citalopram is used alone.

Participants will undergo a 16-week double-blind trial and be randomized to one of three groups: those receiving both MPH and citalopram, those receiving MPH and a placebo, or those receiving citalopram and a placebo. In addition to assessing clinical outcomes, the study will also look at relationships between selected dopamine and serotonin-related genes, mood, cognitive symptoms, and treatment response. This may provide a more complete understanding of the neurobiology of depression in older adults.

Given the significant risk of relapse, frailty, and suicide associated with untreated residual depressive symptoms in older adults, this study could be pivotal. If successful, it could herald a new, potentially more effective approach to treating depression in older adults and improving their quality of life. As the world's population ages, this research could profoundly impact our health systems, social structures, and understanding of mental health in older people.

8.6.2 How is the study designed?

Masking: Quadruple (Participant, Care Provider, Investigator, Outcomes Assessor)

Participant Group/Arm	**Intervention/Treatment**
Active Comparator: 1 - Citalopram and placebo Participants will take a combination	Drug: Citalopram The citalopram dose ranged from 20 to 60 mg daily before the FDA issued a boxed warning in 2011, restricting it to -40 mg. Participants will begin taking a 20 mg capsule once daily for four weeks, with the dosage being increased or decreased based on the participant's response to the drug or side effect profile. Participants will remain on their assigned citalopram dose, which will be increased after week 4 if Clinical Global Impressions (CGI) scores are greater than 2 until the end of the treatment.

Continued on next page

Table 8.3 – continued from previous page

Participant Group/Arm	Intervention/Treatment
of citalopram and placebo for 16 weeks	Other names: Celexa Drug: Placebo The placebo tablets are taken with the active medication. Participants will start with 1 capsule twice daily, increasing to maximum of 16 capsules twice daily for methylphenidate and 1-3 capsules for citalopram. After visit 11, the placebo dose will be gradually reduced over two weeks until participants no longer take any tablets.
Active Comparator: 2 - Methylphenidate and placebo Participants will take a combination of methylphenidate and placebo for 16 weeks	Drug: Methylphenidate (MPH) The dosage of MPH is 5 to 40 mg daily. Participants initially takes 1 capsule (2.5 mg) twice daily, which is increased to a maximum of up to 8 capsules twice daily. After Visit 11, MPH dosage will be gradually reduced over 2 weeks until participants are no longer taking it. Other Names: Ritalin Drug: Placebo The placebo tablets are taken with the active medication. Participants will start with 1 capsule twice daily, increasing to maximum of 16 capsules twice daily for methylphenidate and 1-3 capsules for citalopram. After visit 11, the placebo dose will be gradually reduced over two weeks until participants no longer take any tablets.

Continued on next page

Table 8.3 – continued from previous page

Participant Group/Arm	Intervention/Treatment
Active Comparator: 3 - Methylphenidate and placebo Participants will take a combination of citalopram and placebo for 16 weeks	Drug: Citalopram The citalopram dose ranged from 20 to 60 mg daily before the FDA issued a boxed warning in 2011, restricting it to -40 mg. Participants will begin taking a 20 mg capsule once daily for four weeks, with the dosage being increased or decreased based on the participant's response to the drug or side effect profile. Participants will remain on their assigned citalopram dose, which will be increased after week 4 if Clinical Global Impressions (CGI) scores are greater than 2 until the end of the treatment. Other names: Celexa Drug: Methylphenidate (MPH) The dosage of MPH is 5 to 40 mg daily. Participants initially takes 1 capsule (2.5 mg) twice daily, which is increased to a maximum of up to 8 capsules twice daily. After Visit 11, MPH dosage will be gradually reduced over 2 weeks until participants are no longer taking it. Other Names: Ritalin

Table 8.3: Arms and Interventions of Citalopram & Methylphenidate

9 Future Research Test Plan

9.1 A Research Test Plan

This study designed ISPE-RCT for randomized controlled trials. However, we didn't test the user acceptance due to time and recruitment limitations. We need a research test to test if ISPE-RCT is user-friendly to those research teams without coding experience. We designed a research test involving recruitment, test questions, and measurements.

9.1.1 Brief Description

Since the adoption among medical professionals without coding experience remains unexplored, this research aims to assess how quickly medical participants without coding backgrounds can learn to use ISPE-RCT through a 30-minute tutorial. By focusing on translating ISPE-RCT language tasks and implementing a clinical trial using ISPE-RCT, this study will provide essential insights into the user acceptance of this tool in the medical field.

9.1.2 Recruitment

Criteria

Participants must come from the medical area and have no coding experience. There is no gender or age limitation.

Method

Recruiting participants will start in a clinical trial, as there may be a further test based on this test's results. If the clinical team members are insufficient, we'll recruit through medical institutions, universities, and professional networks.

9.1.3 Test Plan

Tutorial Phase: We will conduct a 30-minute tutorial, including a basic introduction to ISPE-RCT, an overview of its syntax and semantics, and a walkthrough of ISPE-RCT's client-side User Interface (UI).
Testing Phase: We will assess participants through the following test questions:

Translating ISPE-RCT Language Tasks

This part gives the participants background and some ISPE-RCT sentences and asks them to translate them into the intervention. This test aims to assess the understanding of ISPE-RCT language, emphasizing the design of actions. Following are some sample example questions:

- giveDose to group1
- after 3 hours giveDose to group1
- every 1 day giveDose to group2
- monitor adverseEvent for ALL
- collectQuestionnaire for ALL
- .....

Implementing a Clinical Trial

This test will give participants a scenario (for example, a completed clinical trial) and ask them to use ISPE-RCT language to implement this trial. This test evaluates the ability to apply ISPE-RCT language in real-world scenarios.

9.1.4 Measurements

We will mark participants' answers according to how many translation tasks they complete correctly and how well they perform in the implementation task. The weighting of the translating and implementing tasks should not be half-half. We should set the weighting based on the difficulty of each question. The sample marking scheme is as follows:

Scoring:

- **Excellent (90-100%):** Comprehensive understanding and application of ISPE-RCT.
- **Good (70-89%):** Proficient use of ISPE-RCT with minor errors.
- **Satisfactory (50-69%):** Basic understanding with significant room for improvement.
- **Poor (0-49%):**Lack of understanding and/or incorrect application of ISPE-RCT.

User Acceptance Levels:

- **High Acceptance:** Scores in the Excellent or Good range, indicating favorable perception and ease of learning.
- **Moderate Acceptance:** Scores in the Satisfactory range, suggesting further training is needed.
- **Low Acceptance:** Scores in the Poor range, indicating significant barriers to learning or accepting ISPE-RCT.

9.1.5 Summary

This test need to collect scores from the participants and use analytical tools to evaluate this data. We will then conclude whether the participants have grasped the language of the ISPE-RCT after the 30-minute tutorial.

This research test will provide valuable insights into the learnability and acceptance of ISPE-RCT among medical professionals without coding experience. The outcomes may guide future educational strategies, system enhancements, and broader implementation in the medical community. It is an essential step towards ensuring that ISPE-RCT's innovative capabilities are accessible and valuable to the very professionals it aims to serve.

9.1.6 The performance of ISPE-RCT in an RCT

The performance of ISPE-RCT in an RCT is also an important aspect we want to consider. If the user acceptance is around moderate to high, we can apply ISPE-RCT to a real RCT and test its performance. So the following is an additional phase of this test.

Introduction

If the user acceptance of ISPE-RCT is found to be moderate to high, the next critical step is to evaluate its performance in a real Randomized Controlled Trial (RCT). This phase aims to assess how efficiently ISPE-RCT can be implemented in an actual clinical setting and determine its impact on the quality and effectiveness of the RCT process.

Criteria for Performance Testing

User Acceptance Requirement	Participants must have scored in the initial test phase in the Excellent, Good, or Satisfactory range.
Clinical Trial Selection	A suitable ongoing or upcoming RCT should be chosen for this phase.

Table 9.1: Criteria for Performance Testing

Performance Testing Procedures

Participants will use ISPE-RCT to design, manage, or evaluate a part or whole of the selected RCT. Regular monitoring and support will be provided to track the progress and identify potential challenges or errors. Moreover, we need continuous feedback from the participants, other involved medical professionals, and possibly even patients will be collected.

Measurements

We will evaluate ISPE-RCT in efficiency, quality, and satisfaction metrics. The details of these metrics are as follows:

Time Taken	Measure the time taken to complete tasks using ISPE-RCT as compared to traditional methods.
Ease of Integration	Assess how seamlessly ISPE-RCT can be integrated into existing RCT workflows.

Table 9.2: Efficiency Metrics

Summary

This additional phase enhances the research test by transitioning from simulated scenarios to real-world applications. The performance of ISPE-RCT in an actual RCT will provide comprehensive insights into its efficiency, quality, and satisfaction levels. The findings can further inform strategic decisions, training approaches, and development directions to foster ISPE-RCT's success in the broader medical field.

Error Rates	Evaluate the frequency and types of errors in using ISPE-RCT.
Compliance and Standards	Assess how well the ISPE-RCT implementation aligns with the clinical and regulatory standards. Moreover, we'd like to assess how well the SCCharts models reduce the failures in clinical trials.

Table 9.3: Quality Metrics

User Satisfaction	Gauge the satisfaction level of the participants using ISPE-RCT in a real-world context.
Participants Experience	Assess the impact of ISPE-RCT on participants experience, including participation, ease, understanding, and comfort.

Table 9.4: Satisfaction Metrics

Works cited

[1] A. M. Lilienfeld, "Ceteris paribus: The evolution of the clinical trial," *Bulletin of the History of Medicine*, vol. 56, no. 1, pp. 1–18, 1982.

[2] J. P. Bull, "The historical development of clinical therapeutic trials," *Journal of chronic diseases*, vol. 10, no. 3, pp. 218–248, 1959.

[3] NIH, *Nih clinical research trials and you*, https://www.nih.gov/health-information/nih-clinical-research-trials-you/basics, Last accessed on 2022-02-12.

[4] L. M. Friedman, C. D. Furberg, D. L. DeMets, D. M. Reboussin, and C. B. Granger, *Fundamentals of clinical trials*. Springer, 2015.

[5] FDA, *The drug development process*, https://www.fda.gov/patients/learn-about-drug-and-device-approvals/drug-development-process, Last accessed on 2022-02-12, 2018.

[6] MedSafe, *Medicines approval process*, https://www.medsafe.govt.nz/medicines/regulatory-approval-process.asp, Last accessed on 2022-02-12, 2019.

[7] C. Buoen, O. J. Bjerrum, and M. S. Thomsen, "How first-time-in-human studies are being performed: A survey of phase i dose-escalation trials in healthy volunteers published between 1995 and 2004," *The Journal of Clinical Pharmacology*, vol. 45, no. 10, pp. 1123–1136, 2005.

[8] S. Piantadosi, *Clinical trials: a methodologic perspective*. John Wiley & Sons, 2017.

[9] M. Hay, D. W. Thomas, J. L. Craighead, C. Economides, and J. Rosenthal, "Clinical development success rates for investigational drugs," *Nature biotechnology*, vol. 32, no. 1, pp. 40–51, 2014.

[10] H. Ledford, "Translational research: 4 ways to fix the clinical trial.," *Nature*, vol. 477, no. 7366, pp. 526–528, 2011.

[11] J. A. DiMasi, L. Feldman, A. Seckler, and A. Wilson, "Trends in risks associated with new drug development: Success rates for investigational drugs," *Clinical Pharmacology & Therapeutics*, vol. 87, no. 3, pp. 272–277, 2010.

[12] E. M. Schimmel, "The hazards of hospitalization," *Annals of internal Medicine*, vol. 60, no. 1, pp. 100–110, 1964.

[13] P. Aspden, J. Wolcott, J. L. Bootman, and L. R. Cronenwett, "Committee on identifying and preventing medication errors," *Preventing medication errors: quality chasm series*, pp. 1269–1272, 2007.

[14] J. N. Brown, S. R. Britnell, A. P. Stivers, and J. L. Cruz, "Focus: Drug development: Medication safety in clinical trials: Role of the pharmacist in optimizing practice, collaboration, and education to reduce errors," *The Yale Journal of Biology and Medicine*, vol. 90, no. 1, p. 125, 2017.

[15] C. P. Landrigan, G. J. Parry, C. B. Bones, A. D. Hackbarth, D. A. Goldmann, and P. J. Sharek, "Temporal trends in rates of patient harm resulting from medical care," *New England Journal of Medicine*, vol. 363, no. 22, pp. 2124–2134, 2010.

[16] R. Hillestad, J. Bigelow, A. Bower, *et al.*, "Can electronic medical record systems transform health care? potential health benefits, savings, and costs," *Health affairs*, vol. 24, no. 5, pp. 1103–1117, 2005.

[17] D. W. Bates, L. L. Leape, and S. Petrycki, "Incidence and preventability of adverse drug events in hospitalized adults," *Journal of general internal medicine*, vol. 8, pp. 289–294, 1993.

[18] D. W. Bates, D. J. Cullen, N. Laird, *et al.*, "Incidence of adverse drug events and potential adverse drug events: Implications for prevention," *Jama*, vol. 274, no. 1, pp. 29–34, 1995.

[19] R. Kaushal, D. W. Bates, C. Landrigan, *et al.*, "Medication errors and adverse drug events in pediatric inpatients," *Jama*, vol. 285, no. 16, pp. 2114–2120, 2001.

[20] A. Bobb, K. Gleason, M. Husch, J. Feinglass, P. R. Yarnold, and G. A. Noskin, "The epidemiology of prescribing errors: The potential impact of computerized prescriber order entry," *Archives of internal medicine*, vol. 164, no. 7, pp. 785–792, 2004.

[21] B. D. Franklin, C. Vincent, M. Schachter, and N. Barber, "The incidence of prescribing errors in hospital inpatients: An overview of the research methods," *Drug safety*, vol. 28, pp. 891–900, 2005.

[22] T. R. Fleming and D. L. DeMets, "Monitoring of clinical trials: Issues and recommendations," *Controlled clinical trials*, vol. 14, no. 3, pp. 183–197, 1993.

[23] M. Cvach, "Monitor alarm fatigue: An integrative review," *Biomedical instrumentation & technology*, vol. 46, no. 4, pp. 268–277, 2012.

[24] K. M. Gayvert, N. S. Madhukar, and O. Elemento, "A data-driven approach to predicting successes and failures of clinical trials," *Cell chemical biology*, vol. 23, no. 10, pp. 1294–1301, 2016.

[25] H. Gao, J. M. Korn, S. Ferretti, *et al.*, "High-throughput screening using patient-derived tumor xenografts to predict clinical trial drug response," *Nature medicine*, vol. 21, no. 11, pp. 1318–1325, 2015.

[26] F. Zhu, X. X. Li, S. Y. Yang, and Y. Z. Chen, "Clinical success of drug targets prospectively predicted by in silico study," *Trends in Pharmacological Sciences*, vol. 39, no. 3, pp. 229–231, 2018.

[27] I. W. Mak, N. Evaniew, and M. Ghert, "Lost in translation: Animal models and clinical trials in cancer treatment," *American journal of translational research*, vol. 6, no. 2, p. 114, 2014.

[28] S. Harrer, P. Shah, B. Antony, and J. Hu, "Artificial intelligence for clinical trial design," *Trends in pharmacological sciences*, vol. 40, no. 8, pp. 577–591, 2019.

[29] P. Aspden and P. Aspden, *Preventing medication errors*. National Acad. Press, 2007.

[30] R. Koppel, J. P. Metlay, A. Cohen, *et al.*, "Role of computerized physician order entry systems in facilitating medication errors," *Jama*, vol. 293, no. 10, pp. 1197–1203, 2005.

[31] D. W. Bates, J. M. Teich, J. Lee, *et al.*, "The impact of computerized physician order entry on medication error prevention," *Journal of the American Medical Informatics Association*, vol. 6, no. 4, pp. 313–321, 1999.

[32] R. Kaushal, K. G. Shojania, and D. W. Bates, "Effects of computerized physician order entry and clinical decision support systems on medication safety: A systematic review," *Archives of internal medicine*, vol. 163, no. 12, pp. 1409–1416, 2003.

[33] G. J. Kuperman and R. F. Gibson, "Computer physician order entry: Benefits, costs, and issues," *Annals of internal medicine*, vol. 139, no. 1, pp. 31–39, 2003.

[34] S. P. Florence, B. Fetscher, M. Flatt, *et al.*, "Pop-pl: A patient-oriented prescription programming language," *ACM SIGPLAN Notices*, vol. 51, no. 3, pp. 131–140, 2015.

[35] D. Klimov and Y. Shahar, "Ialarm: An intelligent alert language for activation, response, and monitoring of medical alerts," in *Process Support and Knowledge Representation in Health Care: AIME 2013 Joint Workshop, KR4HC 2013/ProHealth 2013, Murcia, Spain, June 1, 2013, Revised Selected Papers*, Springer, 2013, pp. 128–142.

[36] Z. Huang, A. Ten Teije, and F. Van Harmelen, "Semanticct: A semantically-enabled system for clinical trials," in *Process Support and Knowledge Representation in Health Care: AIME 2013 Joint Workshop, KR4HC 2013/ProHealth 2013, Murcia, Spain, June 1, 2013, Revised Selected Papers*, Springer, 2013, pp. 11–25.

[37] J. N. Matthews, *Introduction to randomized controlled clinical trials*. Chapman and Hall/CRC, 2006.

[38] A. Hróbjartsson, F. Emanuelsson, A. S. Skou Thomsen, J. Hilden, and S. Brorson, "Bias due to lack of patient blinding in clinical trials. a systematic review of trials randomizing patients to blind and nonblind sub-studies," *International journal of epidemiology*, vol. 43, no. 4, pp. 1272–1283, 2014.

[39] A. Deaton and N. Cartwright, "Understanding and misunderstanding randomized controlled trials," *Social science & medicine*, vol. 210, pp. 2–21, 2018.

[40] W. F. Rosenberger and J. M. Lachin, *Randomization in clinical trials: theory and practice*. John Wiley & Sons, 2015.

[41] D. Nunan, C. Heneghan, and E. A. Spencer, "Catalogue of bias: Allocation bias," *BMJ Evidence-Based Medicine*, vol. 23, no. 1, p. 20, 2018.

[42] A. Paludan-Müller, D. R. Teindl Laursen, and A. Hróbjartsson, "Mechanisms and direction of allocation bias in randomised clinical trials," *BMC medical research methodology*, vol. 16, no. 1, pp. 1–10, 2016.

[43] J. Fletcher, "Allocation bias," *BMJ*, vol. 338, 2009.

[44] J. A. Kobashigawa, *Us donor heart allocation bias for men over women? a closer look*, 2014.

[45] I. Hyodo, "Patient heterogeneity and allocation bias: How should they be reported in clinical trials of chemotherapy for advanced gastric cancer?" *Gastric Cancer*, vol. 15, no. 2, pp. 115–117, 2012.

[46] P. Sedgwick, "Selection bias versus allocation bias," *BMJ*, vol. 346, 2013.

[47] S. H. Ng, "Biases in reward allocation resulting from personal status, group status, and allocation procedure," *Australian Journal of Psychology*, vol. 37, no. 3, pp. 297–307, 1985.

[48] E. M. Hsich, R. C. Starling, E. H. Blackstone, *et al.*, "Does the unos heart transplant allocation system favor men over women?" *JACC: Heart Failure*, vol. 2, no. 4, pp. 347–355, 2014.

[49] P. Meier, "Stratification in the design of a clinical trial," *Controlled clinical trials*, vol. 1, no. 4, pp. 355–361, 1981.

[50] R. J. Murphy, R. L. Sumner, W. Evans, *et al.*, "Mdlsd: Study protocol for a randomised, double-masked, placebo-controlled trial of repeated microdoses of lsd in healthy volunteers," *Trials*, vol. 22, pp. 1–15, 2021.

[51] M. P. Jensen, C. Chen, and A. M. Brugger, "Interpretation of visual analog scale ratings and change scores: A reanalysis of two clinical trials of postoperative pain," *The Journal of pain*, vol. 4, no. 7, pp. 407–414, 2003.

[52] B. Colagiuri, L. A. Schenk, M. D. Kessler, S. G. Dorsey, and L. Colloca, "The placebo effect: From concepts to genes," *Neuroscience*, vol. 307, pp. 171–190, 2015.

[53] T. J. Kaptchuk and F. G. Miller, "Placebo effects in medicine," *N Engl J Med*, vol. 373, no. 1, pp. 8–9, 2015.

[54] R. B. D'Agostino, "Estimating treatment effects using observational data," *Jama*, vol. 297, no. 3, pp. 314–316, 2007.

[55] G. D. Pozgar, *Legal and ethical issues for health professionals*. Jones & Bartlett Learning, 2019.

[56] C. N. Schmickl, M. Li, G. Li, *et al.*, "The accuracy and efficiency of electronic screening for recruitment into a clinical trial on copd," *Respiratory medicine*, vol. 105, no. 10, pp. 1501–1506, 2011.

[57] R. Simon and A. Maitournam, "Evaluating the efficiency of targeted designs for randomized clinical trials," *Clinical Cancer Research*, vol. 10, no. 20, pp. 6759–6763, 2004.

[58] S. R. Thadani, C. Weng, J. T. Bigger, J. F. Ennever, and D. Wajngurt, "Electronic screening improves efficiency in clinical trial recruitment," *Journal of the American Medical Informatics Association*, vol. 16, no. 6, pp. 869–873, 2009.

[59] Rossum, Guido Van, *The history of python: A brief timeline of python*, https://python-history.blogspot.com/2009/01/brief-timeline-of-python.html, Last accessed on 2022-12-02, 2009.

[60] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in science & engineering*, vol. 9, no. 03, pp. 90–95, 2007.

[61] R. Ihaka and R. Gentleman, "R: A language for data analysis and graphics," *Journal of computational and graphical statistics*, vol. 5, no. 3, pp. 299–314, 1996.

[62] D. Uschner, D. Schindler, R.-D. Hilgers, and N. Heussen, "Randomizer: An r package for the assessment and implementation of randomization in clinical trials," *Journal of Statistical Software*, vol. 85, pp. 1–22, 2018.

[63] *Sas company history*, https://www.sas.com/en_hk/company-information/history.html, Last accessed on 2022-12-02.

[64] *Advantages and disadvantages of sas.* https://www.javatpoint.com/sas-tutorial, Last accessed on 2022-12-02.

[65] J.-J. Li, L.-S. Hou, P. Zhu, X.-D. Du, and C.-R. Zhu, "Sample size calculation using sas for phasein two-stage clinical trials of anti-tumor drugs," *Sichuan da xue xue bao. Yi xue ban= Journal of Sichuan University. Medical Science Edition*, vol. 48, no. 4, pp. 600–604, 2017.

[66] A. Dmitrienko, *Analysis of clinical trials using SAS: a practical guide*. SAS institute, 2017.

[67] Y. Handelsman, J. I. Mechanick, L. Blonde, *et al.*, "American association of clinical endocrinologists medical guidelines for clinical practice for developing a diabetes mellitus comprehensive care plan: Executive summary," *Endocrine practice*, vol. 17, no. 2, pp. 287–302, 2011.

[68] D. W. Curtis, E. J. Pino, J. M. Bailey, *et al.*, "Smart—an integrated wireless system for monitoring unattended patients," *Journal of the American Medical Informatics Association*, vol. 15, no. 1, pp. 44–53, 2008.

[69] O. Chipara, C. Brooks, S. Bhattacharya, *et al.*, "Reliable real-time clinical monitoring using sensor network technology," in *AMIA Annual Symposium Proceedings*, American Medical Informatics Association, vol. 2009, 2009, p. 103.

[70] S. Patel, H. Park, P. Bonato, L. Chan, and M. Rodgers, "A review of wearable sensors and systems with application in rehabilitation," *Journal of neuroengineering and rehabilitation*, vol. 9, no. 1, pp. 1–17, 2012.

[71] Y. Shahar, "A framework for knowledge-based temporal abstraction," *Artificial intelligence*, vol. 90, no. 1-2, pp. 79–133, 1997.

[72] D. Boaz and Y. Shahar, "Idan: A distributed temporal-abstraction mediator for medical databases," in *Artificial Intelligence in Medicine: 9th Conference on Artificial Intelligence, in Medicine in Europe, AIME 2003, Protaras, Cyprus, October 18-22, 2003. Proceedings 9*, Springer, 2003, pp. 21–30.

[73] T. A. Henzinger and J. Sifakis, "The embedded systems design challenge," in *FM 2006: Formal Methods: 14th International Symposium on Formal Methods, Hamilton, Canada, August 21-27, 2006. Proceedings 14*, Springer, 2006, pp. 1–15.

[74] D. Harel and A. Pnueli, *On the development of reactive systems*. Springer, 1985.

[75] P. A. Laplante *et al.*, *Real-time systems design and analysis*. Wiley New York, 2004.

[76] A. Gamatié, *Designing embedded systems with the Signal programming language: synchronous, reactive specification*. Springer Science & Business Media, 2009.

[77] N. Halbwachs, *Synchronous programming of reactive systems*. Springer Science & Business Media, 2013, vol. 215.

[78] A. Benveniste, P. Caspi, S. A. Edwards, N. Halbwachs, P. Le Guernic, and R. De Simone, "The synchronous languages 12 years later," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 64–83, 2003.

[79] D. Harel, "Statecharts: A visual formalism for complex systems," *Science of computer programming*, vol. 8, no. 3, pp. 231–274, 1987.

[80] C. André, "Syncharts: A visual representation of reactive behaviors," *I3S, Sophia-Antipolis, France, Tech. Rep. RR*, pp. 95–52, 1996.

[81] R. Von Hanxleden, B. Duderstadt, C. Motika, *et al.*, "Scharts: Sequentially constructive statecharts for safety-critical applications: Hw/sw-synthesis for a conservative extension of synchronous statecharts," in *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2014, pp. 372–383.

[82] B. Farrell, S. Kenyon, and H. Shakur, "Managing clinical trials," *Trials*, vol. 11, no. 1, pp. 1–6, 2010.

[83] E. L. Eisenstein, R. Collins, B. S. Cracknell, *et al.*, "Sensible approaches for reducing clinical trial costs," *Clinical Trials*, vol. 5, no. 1, pp. 75–84, 2008.

[84] D. B. Hunninghake, C. A. Darby, and J. L. Probstfield, "Recruitment experience in clinical trials: Literature summary and annotated bibliography," *Controlled clinical trials*, vol. 8, no. 4, pp. 6–30, 1987.

[85] M. H. Trivedi, P. J. McGrath, M. Fava, *et al.*, "Establishing moderators and biosignatures of antidepressant response in clinical care (embarc): Rationale and design," *Journal of psychiatric research*, vol. 78, pp. 11–23, 2016.

[86] C. A. Webb, M. H. Trivedi, Z. D. Cohen, *et al.*, "Personalized prediction of antidepressant v. placebo response: Evidence from the embarc study," *Psychological medicine*, vol. 49, no. 7, pp. 1118–1127, 2019.

[87] D. A. Pizzagalli, C. A. Webb, D. G. Dillon, *et al.*, "Pretreatment rostral anterior cingulate cortex theta activity in relation to symptom improvement in depression: A randomized clinical trial," *JAMA psychiatry*, vol. 75, no. 6, pp. 547–554, 2018.

[88] T. Greenberg, H. W. Chase, J. R. Almeida, *et al.*, "Moderation of the relationship between reward expectancy and prediction error-related ventral striatal reactivity by anhedonia in unmedicated major depressive disorder: Findings from the embarc study," *American Journal of Psychiatry*, vol. 172, no. 9, pp. 881–891, 2015.

[89] W. H. Self, U. Sandkovsky, C. S. Reilly, *et al.*, "Efficacy and safety of two neutralising monoclonal antibody therapies, sotrovimab and brii-196 plus brii-198, for adults hospitalised with covid-19 (tico): A randomised controlled trial," *The Lancet Infectious Diseases*, vol. 22, no. 5, pp. 622–635, 2022.

[90] A.-T. B. S. Group*, "Responses to a neutralizing monoclonal antibody for hospitalized patients with covid-19 according to baseline antibody and antigen levels: A randomized controlled trial," *Annals of internal medicine*, vol. 175, no. 2, pp. 234–243, 2022.

[91] J. Lundgren, B Grund, C. Barkauskas, *et al.*, "Activ-3/tico ly-cov555 study group. a neutralizing monoclonal antibody for hospitalized patients with covid-19," *N Engl J Med*, vol. 384, no. 10, pp. 905–14, 2021.

[92] L. M. Christian, A. S. Young, A. M. Mitchell, *et al.*, "Body weight affects ω-3 polyunsaturated fatty acid (pufa) accumulation in youth following supplementation in post-hoc analyses of a randomized controlled trial," *PLoS One*, vol. 12, no. 4, e0173087, 2017.

[93] A. S. Young, L. E. Arnold, H. L. Wolfson, and M. A. Fristad, "Psychoeducational psychotherapy and omega-3 supplementation improve co-occurring behavioral problems in youth with depression: Results from a pilot rct," *Journal of abnormal child psychology*, vol. 45, pp. 1025–1037, 2017.

[94] Alexander Schulz-Rosengarten, *Kieler compiler*, https://rtsys.informatik.uni-kiel.de/confluence/display/KIELER/Kieler+Compiler, Last accessed on 2023-04-28, 2018.

[95] Alexander Schulz-Rosengarten, *Downloads - kieler compiler command-line interface*, https://rtsys.informatik.uni-kiel.de/confluence/display/KIELER/Downloads+-+KIELER+Compiler+Command-Line+Interface, Last accessed on 2023-04-28, 2021.

[96] M. Bergström, A. Grahnen, and B. Långström, "Positron emission tomography microdosing: A new concept with application in tracer and early clinical drug development," *European journal of clinical pharmacology*, vol. 59, pp. 357–366, 2003.

[97] K. P. Kuypers, L. Ng, D. Erritzoe, *et al.*, "Microdosing psychedelics: More questions than answers? an overview and suggestions for future research," *Journal of Psychopharmacology*, vol. 33, no. 9, pp. 1039–1057, 2019.

[98] T. Lea, N. Amada, H. Jungaberle, H. Schecke, and M. Klein, "Microdosing psychedelics: Motivations, subjective effects and harm reduction," *International Journal of Drug Policy*, vol. 75, p. 102 600, 2020.

[99] University of North Carolina, Chapel Hill, *Itbs for adolescent depression: An open label study evaluating safety and efficacy of treatment*, https://clinicaltrials.gov/study/NCT04485455?aggFilters=results:with,status:com&viewType=Table&cond=depression&rank=9, Last accessed on 2023-06-28, 2021.

[100] Weill Medical College of Cornell University, *The feasibility of engage therapy with video support for homebound older adults*, https://clinicaltrials.gov/study/NCT05346055?aggFilters=results:with,status:com&viewType=Table&page=3&rank=25, Last accessed on 2023-06-30, 2023.

[101] C. Galo, N. Benda, I. O. Rollandi, S. Czaja, M. Ceruso, and J. A. Sirey, "A pilot assessment of a tablet-based intervention for homebound or socially isolated older adults with depression," *Innovation in Aging*, vol. 6, no. Supplement_1, pp. 878–878, 2022.

[102] The United States governemnt, *Clinicaltrials.gov: A place to learn about clinical studies from around the world*, https://clinicaltrials.gov/, Last accessed on 2023-07-01, 2023.

[103] Susanna Naggie, MD, Duke University (Responsible Party), *Activ-6: Covid-19 study of repurposed medications - arm c (fluticasone)*, https://clinicaltrials.gov/study/NCT05736874?aggFilters=results:with,status:com&viewType=Table&rank=3, Last accessed on 2023-07-01, 2023.

[104] University of North Carolina, Chapel Hill, *Coronavirus disease 2019 (covid-19) antibody plasma research study in hospitalized patients (unc ccp rct)*, https://clinicaltrials.gov/study/NCT04524507?aggFilters=results:with,status:com&viewType=Table&cond=Covid19&page=4&rank=40, Last accessed on 2023-07-01, 2023.

[105] L. A. Bartelt, A. J. Markmann, B. Nelson, *et al.*, "Outcomes of convalescent plasma with defined high versus lower neutralizing antibody titers against sars-cov-2 among hospitalized patients: Coronavirus inactivating plasma (covip) study," *Mbio*, vol. 13, no. 5, e01751–22, 2022.

[106] Wei Jiang, Duke University (Responsible Party), *Omega 3 for treatment of depression in patients with heart failure (ocean)*, https://clinicaltrials.gov/study/NCT02057406?aggFilters=results:with,status:com&viewType=Table&cond=depression&page=8&rank=76, Last accessed on 2023-07-01, 2018.

[107] W. Jiang, D. J. Whellan, K. F. Adams, *et al.*, "Long-chain omega-3 fatty acid supplements in depressed heart failure patients: Results of the ocean trial," *JACC: Heart Failure*, vol. 6, no. 10, pp. 833–843, 2018.

[108] K. M. Appleton, P. D. Voyias, H. M. Sallis, *et al.*, "Omega-3 fatty acids for depression in adults," *Cochrane Database of Systematic Reviews*, no. 11, 2021.

[109] Helen Lavretsky, MD, University of California, Los Angeles (Responsible Party), *Effectiveness of methylphenidate in improving cognition and function in older adults with depression*, https://clinicaltrials.gov/study/NCT00602290?aggFilters=results:with,status:com&viewType=Table&cond=depression&page=4&rank=37, Last accessed on 2023-07-01, 2018.

[110] B. Schneider, L. Ercoli, P. Siddarth, and H. Lavretsky, "Vascular burden and cognitive functioning in depressed older adults," *The American Journal of Geriatric Psychiatry*, vol. 20, no. 8, pp. 673–681, 2012.

[111] H. A. Eyre, A. Eskin, S. F. Nelson, *et al.*, "Genomic predictors of remission to antidepressant treatment in geriatric depression using genome-wide expression analyses: A pilot study," *International journal of geriatric psychiatry*, vol. 31, no. 5, pp. 510–517, 2016.

[112] Oracle Team, *Interface operatingsystemmxbean*, https://docs.oracle.com/en/java/javase/11/docs/api/jdk.management/com/sun/management/OperatingSystemMXBean.html, Last accessed on 2023-08-22, 2023.

[113] G. Kereszty and R. van Nieuwpoort, "Monitoring and management support for ibis runtime systems and grid applications," Ph.D. dissertation, M. Sc. thesis, 2007, to be published.

[114] D. DONSEZ, "Jmx java management extension,"

[115] J. Friesen, "Monitoring and management," *Beginning Java™ SE 6 Platform: From Novice to Professional*, pp. 221–252, 2007.

[116] R. S. Chowhan, A. Mishra, and A. Mathur, "Aglet and kerrighed as a tool for load balancing and scheduling in distributed environment," in *2016 International Conference on Recent Advances and Innovations in Engineering (ICRAIE)*, IEEE, 2016, pp. 1–6.

[117] dbwiddis, *Github open source of oshi*, https://github.com/oshi/oshi/tree/master, Last accessed on 2023-07-18, 2023.

[118] R. Firment, "Monitoring support for manta flow agent in cloud-based architecture," 2022.

[119] W. Ding, F. Luo, L. Han, C. Gu, H. Lu, and J. Fuentes, "Adaptive virtual machine consolidation framework based on performance-to-power ratio in cloud data centers," *Future Generation Computer Systems*, vol. 111, pp. 254–270, 2020.

[120] A. A. Silva, J. Rocheteau, C.-L. Pihery, and P. Mabit, "Measuring green software enginnering monitoring software energy consumption,"

[121] P. Hsia, D. Kung, and C. Sell, "Software requirements and acceptance testing," *Annals of software Engineering*, vol. 3, no. 1, pp. 291–317, 1997.

[122] M. L. Hutcheson, *Software testing fundamentals: Methods and metrics*. John Wiley & Sons, 2003.

[123] B. Homès, *Fundamentals of software testing*. John Wiley & Sons, 2013.

[124] G. Chen, R Shetty, M Kandemir, N. Vijaykrishnan, M. J. Irwin, and M. Wolczko, "Tuning garbage collection for reducing memory system energy in an embedded java environment," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 1, no. 1, pp. 27–55, 2002.

[125] G. Chen, R Shetty, M. Kandemir, N. Vijaykrishnan, M. J. Irwin, and M. Wolczko, "Tuning garbage collection in an embedded java environment," in *Proceedings Eighth International Symposium on High Performance Computer Architecture*, IEEE, 2002, pp. 92–103.

[126] A. Soni and V. Ranga, "Api features individualizing of web services: Rest and soap," *International Journal of Innovative Technology and Exploring Engineering*, vol. 8, no. 9, pp. 664–671, 2019.

[127] H. Mei and L. Zhang, "A framework for testing web services and its supporting tool," in *IEEE International Workshop on Service-Oriented System Engineering (SOSE'05)*, IEEE, 2005, pp. 199–206.

[128] G. S. Alexopoulos, P. J. Raue, S. Banerjee, *et al.*, "Comparing the streamlined psychotherapy "engage" with problem-solving therapy in late-life major depression. a randomized clinical trial," *Molecular psychiatry*, vol. 26, no. 9, pp. 5180–5189, 2021.

[129] G. S. Alexopoulos, P. J. Raue, F. Gunning, *et al.*, ""engage" therapy: Behavioral activation and improvement of late-life major depression," *The American Journal of Geriatric Psychiatry*, vol. 24, no. 4, pp. 320–326, 2016.

[130] S. J. Czaja, W. R. Boot, N. Charness, W. A. Rogers, and J. Sharit, "Improving social support for older adults through technology: Findings from the prism randomized controlled trial," *The Gerontologist*, vol. 58, no. 3, pp. 467–477, 2018.