UNIVERSITY OF LEEDS

Ontology-Supported Scaffolding for System Safety Analysis

Paul S Brown

Submitted in accordance with the requirements for the degree of Doctor of Philosophy

The University of Leeds

Faculty of Engineering

School of Computing

April 2022

Intellectual Property

The candidate confirms that the work submitted is his/her own and that appropriate credit has been given where reference has been made to the work of others.

This copy has been supplied on the understanding that it is copyright material and that no quotation from the thesis may be published without proper acknowledgement.

"via, veritas, vita"

Soli Deo Gloria

Acknowledgements

I would like to acknowledge and thank several people who have helped with this research. First and foremost I must express my deepest gratitude to my supervisors: Vania Dimitrova, Anthony G. Cohn, and Glen Hart. They've spent many hours talking, reading, and advising. The expertise provided by Glen and Tony has been indispensable. In addition, Vania's mentorship has been invaluable. Thank you all for providing me with this opportunity, sticking with me through all four-and-a-half years of it, and aiding me in learning so much. What I've learned during the PhD has utility not only for research and career, but also for argument formulation and critical thinking.

I must also thank my family who've been awaiting the completion of this research for far too long. Now it's finished, we'll get on with all those plans we've been making. Thank you for your patience, listening to me talking in the babblish language that goes with research, and your insistance that I finish this before moving on with all those plans we've made.

Many thanks to Anne Ogborn, who patiently introduced me to Prolog. And many thanks to Paulo Moura, who carried on my Prolog education, patiently introduced me to Logtalk, pointed me towards how to use it to solve the problems I was encountering with Prolog, and has become both a friend and colleague. That the software runs is only due to the generosity of Paulo and his teaching; I began this research knowing no logic programming. I look forward to many more hours Logtalking on future projects.

I was also fortunate to have the opportunity to discuss my research with several kind and patient academics beyond my wonderful supervisors. I must thank "the team" who met with me from the DSTL who suggested that I focus on STAMP. Thanks to Roberto Ferrario who took time to introduce me to the philosophical side of ontology and discuss my ontological-plans. And many thanks to Brandon Bennett, who patiently entertained me when I dropped by his office

at random times to chat about all things logical. During the Covid-19 enforced lockdown, these chats with Brandon were the thing I missed the most about being on campus.

My studies were supported by the UKRI. Additional support was provided by the Defence Science and Technology Laboratory (DSTL). Many thanks for your support, I wouldn't have been able to conduct this research without it.

Ultimately, as in all things, Soli Deo Gloria.

Some of the work in this thesis has been published prior to submission.

- The concept of using Contingent Scaffolding for System Safety analysis via ontology and Situation Calculus was initially proposed in the Doctorial Concortium paper:

Paul S Brown, Anthony G Cohn, Glen Hart, and Vania Dimitrova (2020). "Contingent Scaffolding for System Safety Analysis". In: *International Conference on Artificial Intelligence in Education*. Springer, pp. 395–399

- An earlier version of **Chapter 5** was presented in:

Paul S Brown, Vania Dimitrova, Glen Hart, Anthony G Cohn, and Paulo Moura (2021). "Refactoring the Whitby Intelligent Tutoring System for Clean Architecture". In: *Theory and Practice of Logic Programming* 21.6, pp. 818–834

- The STAMP ontology from **Chapter 3** with examples has been published for public access:

Paul S. Brown (Apr. 2022). *PaulBrownMagic/STAMP-Ontology: Initial Release*. Version v1.0.0. DOI: 10.5281/zenodo.6489774. URL: https://doi.org/10.5281/zenodo.6489774

Abstract

System Safety Analysis is a valuable task used when trying to ensure that any thing that can be represented with the systems-model does not behave in some manner that is undesirable to the stakeholders in that system. It's a creative task, with no known correct solution, with limited tool support. This thesis investigates the possibility of providing support to analysts undertaking this task through the use of ontology and pedagogy in an artificially intelligent tool.

An ontology to capture the system-model as understood by System-Theoretic Accident Model and Processes (STAMP) was authored, building on an existing set-theoretic representation. This required the authoring of underlying ontology-modules, including one for Control Systems and one to capture sufficient information for use with Situation Calculus. Together these capture information to be used in reasoning about system behaviour. During System Safety Analysis a user extends this ontology to model their system, and the intelligent support tool interprets it to offer its advice.

The intelligent support tool uses Contingent Scaffolding to tailor its support to the user, this pedagogical strategy was chosen as it's been shown to enable the learner to produce a better quality product than they would be capable of alone. Contingent Scaffolding depends upon knowledge of past behaviour of the learner so that interventions can be pitched at the correct level for the learner. Typically ontology authoring tools use a synchronic view of the ontology, and so don't capture the required history. This tool uses Situation Calculus to capture a diachronic view of the ontology such that the history of authorship can be reasoned with to apply the Contingent Scaffolding framework defined herein.

To evaluate the practicability of this approach the ontology and scaffolding were implemented in software. This surfaced an issue with the inability to inverse dependencies in Prolog, which was important to make the tools reuseable and shareable. These were overcome by Protocols

provided in Logtalk. The code was then applied to other domains, such as robotics planning by a third-party, demonstrating generalisability of the intelligent support tool.

A user study was conducted to evaluate the effectiveness of the intelligent support tool, in which novices undertook a System Safety Analysis. The tool was able to effectively provide support where definitions were missed and additional patterns of behaviour were identified that are indicitive of the user needing support.

The thesis makes a number of contributions including: a systems ontology with a focus on capturing hypothetical and realised behaviour, a formal definition of a contingent scaffolding framework that can be used with ill-defined tasks, and the use of dependency inversion in Prolog to enable sharing of libraries. The primary contribution is in the use of a diachronic view of ontology authoring to provide support, which has been successfully exploited and has scope for providing a platform for many more applications.

Contents

List of Figures

List of Tables

Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
BFO	Basic Formal Ontology
CAST	Causal Analysis based on STamp
DCG	Definite Clause Grammer
DL	Description Logic
DOLCE	Descriptive Ontology for Linguistic and Cognitive Engineering
DSL	Domain Specific Language
GFO	General Formal Ontology
GUI	Graphical User Interface
HTML	HyperText Markup Language
ID	IDentity
KB	Knowledge Base
NASA TLX	National Aeronautics and Space Administration Task Load indeX
ODP	Ontology Design Pattern
OSWIN	Ontology-driven Scaffolding with Interactive Nudges
OWL	Web Ontology Language
RDF	Resource Description Framework
SOLID	Single responsibility, Open-closed, Liskov substitution, Interface segregation, and Dependency inversion principles
STAMP	System-Theoretic Accident Model and Process
STPA	System Theoretic Process Analysis
STPA-Sec	STPA for Security
STRIPS	STanford Research Institute Problem Solver
UCA	Unsafe Control Action
UFO	Unified Foundational Ontology
UI	User Interface

Chapter 1

Introduction

The Ariane 5, a satellite payload rocket, took its maiden flight on 4^{th} June 1996. 37 seconds after liftoff, it exploded, incurring a direct cost of ~$370,000,000 (Dowson 1997; Nuseibeh 1997). The immediate cause was tracked down to an overflow error when converting a 64-bit floating point number to a 16-bit integer (Nuseibeh 1997). However, the software was operating exactly to its requirements, which were defined for prior versions of the Ariane and were inappropriate for the Ariane 5 (Dowson 1997). Therefore other causal factors were identified including issues in design, requirements, testing, project and risk management: a systemic process problem (Nuseibeh 1997).

The Therac-25, a software-controlled radiation therapy machine, was in use between June 1985 and January 1987 (Nancy G Leveson and Turner 1993). During that time it massively overdosed at least six people, who were seriously injured and some of whom died. Analysis of the sequence of incidents indicates that they were not fully addressed on each occurrence, resulting in the possibility of further events, which kept occurring (Nancy G Leveson and Turner 1993). However, in systems such as those capable of delivering lethal doses of radiation, it's not sufficient to correct a discovered mistake, such systems need to be safe first time. The Therac-25 was developed from an earlier model, the Therac-6, as was the Therac-20. Informal investigation showed under similar conditions to those identified as one of the causes of several incidents, the Therac-20's mechanical interlocks would result in blown fuses and breakers, whereas the Therac-25's software would deliver the lethal dose of radiation (Nancy G Leveson and Turner 1993).

System Safety Analysis is a high-stakes, high-value activity impacting our lives as we interact

with the myriad of things that can be described as systems with the potential to behave in an undesirable manner. It is also a creative process with no known correct answer or point of completion, which may be undertaken by non-expert analysts. Therefore, there is great value in supporting analysts to produce as high-quality analyses as possible whilst also being a challenging task to support with artificial intelligence.

The impetus behind this research is the idea that ontology can help support System Safety Analysis, based upon ontology providing a shared model between human and machine that both can reason with. To achieve this goal multiple topics are tackled in turn, each with their own questions to address.

Under the topic of ontology:

- Can the STAMP[1] model be satisfactorily capture in an ontology?
- How is it possible to capture sufficient information to reason about hypothetical futures?
- Does the ontology enable sufficient reasoning to provide support to an analyst?

Under the topic of support, which is provided by Contingent Scaffolding:

- Can a formal definition of Contingent Scaffolding be authored that provides flexibility?[2]
- What would a general definition of the levels of support provided be in a digital context?
- Does the additional information in the diachronic[3] view of the ontology authoring process offer any benefit over the synchronic one?
- Is the defined framework for Contingent Scaffolding effective in this creative domain of System Safety Analysis?

Under the topic of software implementation:

- Can the software operate at sufficient speed as to provide perceptually immediate feedback?
- Can the generalisable libraries, such as those for reasoning about situations, ontology authoring, and Contingent Scaffolding be shared as testimony to their being applicable to other domains.

To address these topics and questions research is conducted by first examining the relevant the-

[1] STAMP is the System-Theoretic Accident Model and Process for System Safety Analysis. See: **Section 2.1.2**

[2] Flexibility to support creative processes, such as system-safety analysis, where the correct answers and user actions can't be deduced.

[3] A diachronic view includes historical changes, as opposed to a synchronic view of only a single point in time with no reference to historical changes.

oretical frameworks, formalising them if necessary, or incorporating them if they are already formal. The result of this is the STAMP ontology and the Contingent Scaffolding framework. These theoretical foundations are then validated for practicability by implementing them in software and ensuring that the software can operate at sufficient speed as to provide immediate feedback to the analyst. The final step is to evaluate the efficacy of the whole (ontology, scaffolding, software with user interface) to improving the quality of a System Safety analyst's model, particularly of those who are beginning to learn how to do such an analysis.

The tackling of these questions is of interest to multiple fields, and so contributes to:

- **System Safety Analysis**: in particular the ontological model as well as novel and effective means of supporting analysts. The ontological approach merits consideration by the community as it biases precision.
- **Ontology Authoring**: in particular the argument for a diachronic view of ontology authoring and again a novel and effective means of supporting the author. Taking into account past behaviour of the author(s) when offering support opens up more possibilities for support.
- **Intelligent Tutoring**: in particular the formal definition of Contingent Scaffolding with its novel means of providing flexibility. This formal definition makes it possible to prove interventions will be provided under certain circumstances, rather than merely testing them.
- **Logic Programming**: a solution to the architectural issue prohibiting sharing code that depends on user defined terms. Application of Clean Architecture to Logic Programming.

This thesis is organised into seven chapters, this is the first one. The second chapter provides a background to the domains used in this research and motivates the choices made. It then summarises the plan for the research based upon this review of related works.

In chapter 3, an ontological model suitable for capturing the System Safety Analysis is defined. It's also necessary to explore the potential reasoning such a model could support in order to inform the kind of support that can be provided. A consideration here is to balance an ontologically sound model with the requirement to express the STAMP[4] model in a manner that could be understood by an analyst who is not a knowledge engineer.

[4]See: **Section 2.1.2**

Chapter 4 defines a framework for Contingent Scaffolding that can be applied to ontology authoring, which unbeknownst to the user of the software application is how their model is represented. The Contingent Scaffolding framework is independent of ontology authoring, and so is applied to ontology authoring in this chapter. The application of Contingent Scaffolding for ontology authoring is also independent of the STAMP ontology, and is applied to it in the software application. The goal here is to create a formal definition for Contingent Scaffolding for testing with other domains.

In Chapter 5 the theoretical frameworks and ontology are incorporated into a software application. While authoring the software it was discovered that shortcomings in the standard Prolog language were inconducive to writing shareable code, which was addressed through Dependency Inversion and Logtalk. This is an important goal as the ontology authoring and Contingent Scaffolding frameworks are claimed to be applicable to other domains, therefore making it easy to share encourages third-party verification of this claim.

Chapter 6 reports on the user-study conducted in which 37 students in the School of Computing at University of Leeds were introduced to the System Safety Analysis methodology used[5], provided with a scenario describing a recommender system to analyse, and were observed conducting their analysis with action logs collected. This was undertaken to test if the ontology-driven Contingent Scaffolding framework, when implemented into software, provided any beneficial support to users who are beginning to learn the System Safety Analysis methodology used. The primary focus is on improving model quality, but the impact on learning is also investigated as it is the process by which a beginner gains expertise.

Chapter 7 summarises the main arguments and findings of this thesis. The generalisability and reapplication of the various aspects are discussed. The immediate improvements identified are put forth, as well as longer term goals. As a preliminary investigation into the topic of ontology-driven support for System Safety Analysis, there is much untapped potential that could not be addressed, which is described for future endeavours.

[5] i.e. STPA, which is introduced in **Section 2.1.2**

Chapter 2

Related Work

2.1 System Safety Analysis

System Safety is:

> The discipline that uses systematic engineering and management techniques to aid in making systems safe throughout their life cycles. (Stephans 2012)

Where "Safety" is "freedom from harm" (Stephans 2012, p.11), and "System" is a "composite of people, procedures, and plant and hardware working within a given environment to perform a given task" (Stephans 2012). The "harm" to be free from is the loss of anything of value to the stakeholders (N. Leveson and Thomas 2018, p.16), which occurs in loss events (N. Leveson 2017, p.465). Analysis is one part of System Safety used to identify and understand hazards such that they can be controlled (Stephans 2012). A hazard is some "system state or set of conditions that, together with a particular set of worst-case environment conditions, will lead to a [loss]" (N. Leveson 2017, p.465).

For safety analysis, the scope of the systems considered is restricted to those that can cause harm and that are controllable so that hazards may be controlled. For the system to cause harm it must interact with its environment. The system described can be defined as a dynamic control system. Where "a dynamic system is one whose present output depends on past inputs." (Auslander 1974, p.3) and a control system is "an interconnection of components forming a system configuration that will provide a desired system response" (Dorf and Bishop 2011, p.24). This definition captures interaction with the environment through inputs and outputs as well as the intention

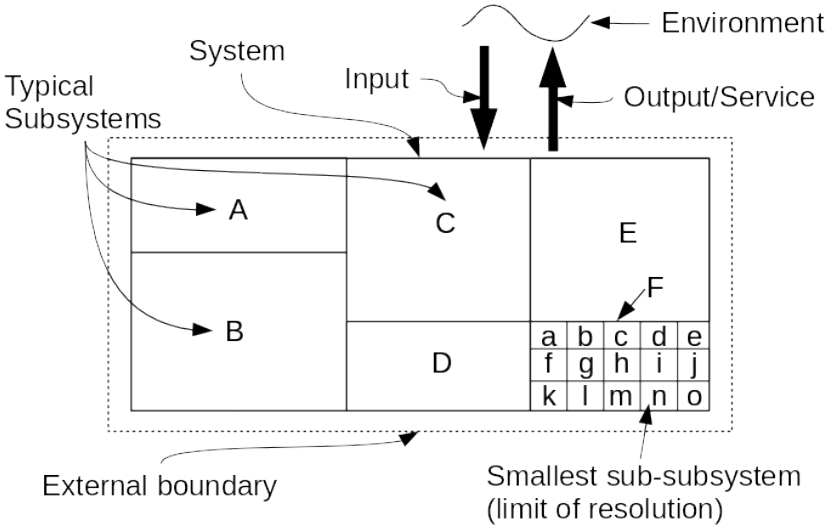

Figure 2.1: System of systems model, adapted from the Fault Tree Handbook (Vesely et al. 1981, p.I-5).

that the system behave in a desirable way, which is to also avoid undesired behaviours as defined by the stakeholders.

Safety cannot be considered without understanding the environment within which the system is placed. Knowing the inputs and outputs to the system is insufficient; 44% of deaths in aviation incidents in 2017 were people on the ground (Shepardson 2018). Therefore a "system of systems" recursive approach is required, shown in **Figure 2.1**. This permits each component to be considered as a system, and the system being analysed to be a component in a broader system (Beer 1979; A. Avižienis et al. 2004; Vesely et al. 1981; Wilson 1984). This allows the analyst to determine the appropriate level of granularity with which to conduct their analysis (Wilson 1984; Vesely et al. 1981).

A model that can describe the process that leads to a loss is required to perform safety analysis in order to control the hazards (which are the conditions enabling a loss to occur). To describe the process it must be able to describe the components that comprise the system and how they are causally related (Zeigler 1976). Therefore the model must also describe the properties or qualities of the components and the qualities that emerge when those components are combined into a system. The qualities are descriptive variables (Zeigler 1976, p.10), meaning they *describe* the situation a system or component is in as they *vary*. In such a model, events are mappings between these situations. This is shown in **Figure 2.2** where circles denote a situation and the arrows between them a mapping from one situation to another.

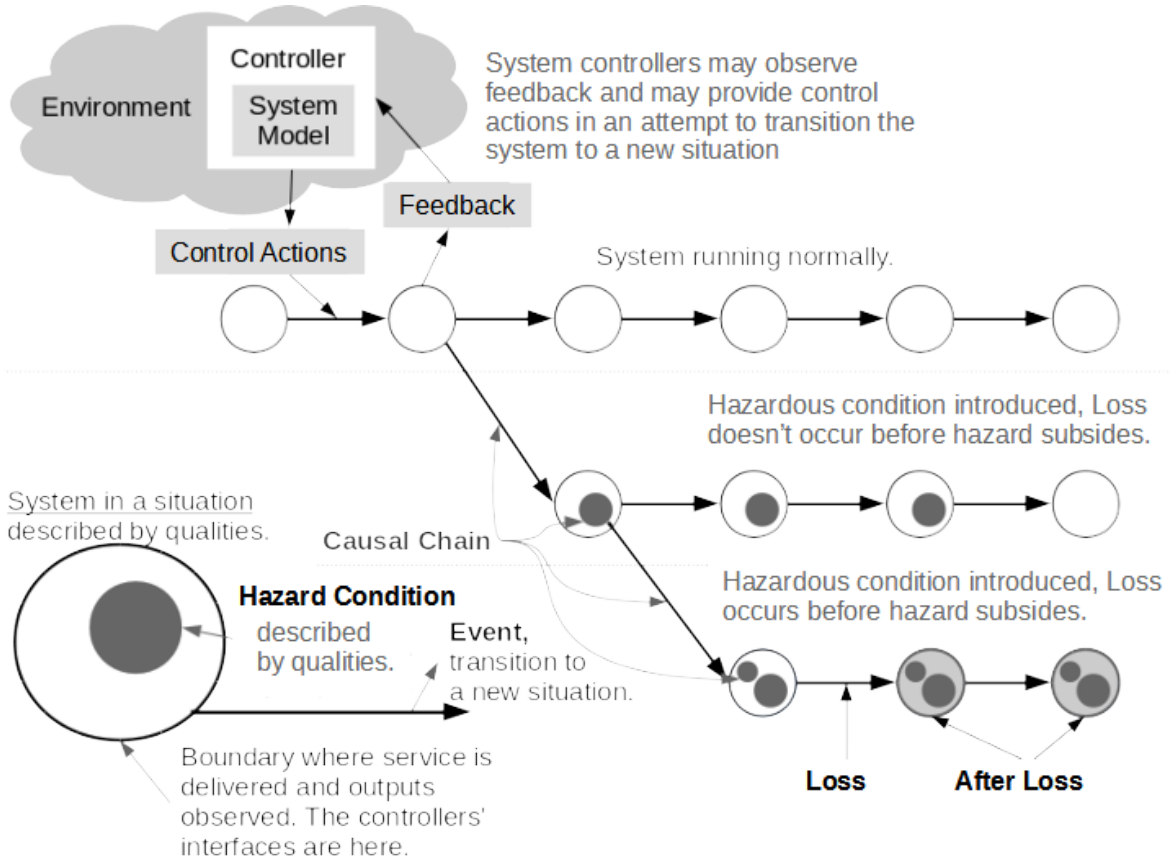

Figure 2.2: System model showing causal chain to system failure

To account for how a system can be controlled the model shown in **Figure 2.2** also includes a controller. The controller accepts output from the system and provides it with input (N. Leveson 2017). Due to the recursive nature of the system definition, a controller can also be considered as a component in the system. A controller can be human or machine, deterministic or stochastic. For the controller to choose inputs to provide they must have access to some model of the system. This model may not be accurate or complete (N. Leveson 2017). Environmental influences that are not controlled, such as wind direction or rainfall, are taken to be the worst-case scenario values and otherwise excluded from the model as they can't be used to intentionally control the hazards: the aim of system safety.

Once the system behaviour in relation to control of hazards is understood, some analysts will proceed to assess the risks associated with each hazard. The System Safety Analysis considered in this thesis is up to and including the identification of unsafe behaviour, which precedes any risk assessment. Therefore it's taken that the mere existence of any hazard is undesirable, regardless of likelihood or severity.

2.1.1 System Safety Analysis Frameworks

The distinction between system safety engineering and reliability engineering must be made prior to introducing the approaches. Safe systems need not be reliable, reliable systems need not be safe, and sometimes safety and reliability constraints can conflict (N. Leveson 2017, p.7). The purpose of reliability engineering is to ensure that the system does not fail to provide the functions for which it was designed. Whereas "System safety is the assurance and management that the system is safe for all people, environment, and equipment." (Bahr 2015, p.4) The two do not necessarily conflict, nor are they necessarily aligned.

The difference is evident considering an example investigation by the AAIB (2018). On 24th September 2017, a three-seat Jodel DR1050-M Excellence aircraft struck three sheep when the pilot experienced control difficulties and baulked the landing. The investigation found no fault or failure in the aircraft, nor did they suggest that the pilot acted inappropriately given the information they had. The system did not fail but two sheep were killed and the aircraft was damaged.

System Safety involves identifying, analysing, and eliminating hazards (Bahr 2015; Vincoli 2006). An older approach from pre-1940s aviation was known as the "fly-fix-fly" approach (Bahr 2015; Stephans 2012), which made learning from errors more explicit than modern approaches that emphasize getting it right first time. During the 1940s-1950s system safety evolved into a discipline to cope with the demand to make systems safe that could not be tested using the fly-fix-fly approach, such as the development of nuclear weapons and space travel (Stephans 2012).

In the 21st century there are multiple System Safety Analysis techniques available. Stephans (2012) describes 20 different techniques, without including STAMP's STPA and CAST as it was published in the same year as N. Leveson (2004) was proposing STAMP as a new model. Each technique frames the safety problem from a particular perspective to accomplish specific goals; the main ones are summarised here:

- **ETBA** (Energy Trace and Barrier Analysis): Frames safety in terms of energy flow, an incident is the unwanted flow of energy (Stephans 2012).
- **FMEA** (Failure Mode and Effects Analysis): Typically for reliability analysis rather than safety. FMEA is a form to record associations between components, failure modes, and

failure effects. It can be used for subsystem and hazard analysis (Stephans 2012).

- **FTA** (Fault Tree Analysis): For reliability analysis rather than safety. Uses a tree for qualitative and quantitative analysis of system failures (Vesely et al. 1981).
- **PET** (Project Evaluation Tree): Provides a graphic checklist to aid in evaluating the system and its parts (Stephans 2012).
- **Change Analysis**: Frames safety as a change problem, and so lists and examines changes systematically, using comparisons between situations, to ensure safety is not effected by changes (Stephans 2012).
- **MORT** (Management Oversight and Risk Tree): Frames safety as a control problem. Uses a tree as a systematic tool for accident investigation to identify the control factors that can be improved to prevent recurrence (Stephans 2012).
- **Event and Causal Factors Charts**: Frames safety as a behavioural problem, inspects the sequence of events that led to an accident. Creates a graphical representation of the causal factors (Stephans 2012).
- **STAMP**: Frames safety as a control problem. Uses a systematic approach to identify the control actions that are potentially unsafe and unsafe processes (N. Leveson 2017).

2.1.2 STAMP and STPA

STAMP is primarily a safety model, it combines systems theory with safety constraints, a hierarchical safety control structure, and process models. At its core is the concept of a controller within a feedback loop that can adjust the state of the system that it controls. Losses are understood by "identifying the safety constraints that were violated and determining why the controls were inadequate in enforcing them." (N. Leveson 2017, p.90) With this model of causality, the occurrence of a loss implies that a controller did not enforce a safety constraint, or that the controller provided appropriate control actions but their results were not manifested.

STAMP can be used in two analysis process methodologies: pre-incident with STPA (**S**ystem-**T**heoretic **P**rocess **A**nalysis), and also post-incident with CAST (**C**ausal **A**nalysis based on **ST**AMP) (N. Leveson 2017). Being grounded in Systems Theory it's amenable to formal modeling, which has been exploited in a set-theoretic representation (Thomas 2013). Although STAMP can be entirely conducted with pencil and paper, the utility of a formal model has been recognised and used to aid analysis via software (Thomas 2013; Gurgel et al. 2015). The set-theoretic model indicates an ontological one can not only provide the same utility, but with

subsumption and additional relations provide a richer model to support additional reasoning tasks. STAMP is chosen as a safety analysis model, rather than a reliability model, which has an existing formal model that can be extended.

STPA is relatively new, gaining results comparable with other methodologies and revealing insights they missed (Fleming et al. 2013; Pawlicki et al. 2016). STPA is the analysis undertaken pre-design, and so imposes the requirement to reason with hypotheticals, classes of things, and branching time. Whereas CAST is undertaken post-incident and so requires instances of things and linear time. Given that instances can be asserted from classes and that linear time is a single branch of time, STPA's requirements on modeling are a superset of CAST's, and so STPA is the primary focus of this research. An brief introduction to STPA is included in **Appendix A**.

2.1.3 Supporting System Safety Analysis

System safety analysis is conducted to understand the behaviour of increasingly complex systems to mitigate or prevent undesirable behaviour. The consequences of inadequate analysis can be catastrophic. Support tools aim to aid in the management of these complex models in order to mitigate the risks associated with mistakes and incompleteness in the models.

Analysts require expert-level knowledge and skills regarding their chosen methodology (such as STPA), chosen domain (such as hydroponic farming), modeling, as well as the system under consideration (such as the Miracle-Gro AeroGarden Farm 24XL: an intelligent hydroponic home system). Given that STPA is an emerging methodology, there are a growing number of people wishing to learn it and its associated model. Expertise regarding the system also cannot be assumed as STPA can be conducted from the design phase, on large systems distributed over teams, and on complex systems requiring expertise in multiple fields. Ideally some kind of support tool would mitigate any lacking expertise.

STPA is an ill-defined task (as defined by Mitrovic and Weerasinghe (2009)) with an ambiguous starting state, an unknown goal state, an advisory non-strict procedure, and no known correct solution. It is an ill-defined domain (as defined by Mitrovic and Weerasinghe (2009)): STPA is generic to all analyses and thus contains incomplete declarative knowledge regarding a particular analysis, including the system under analysis. System safety is an ill-defined problem (as defined by Lynch et al. (2009)), in STPA safety is re-characterised as a control problem, alternative characterisations include Swiss-cheese and dominoes (N. Leveson 2017), others are discussed in

Section 2.1.1. In summary, STPA is "ill-defined", a term used in this text to denote the three kinds of ill-defined here, which limits the support that can be provided. For such ill-defined domains it is necessary to scaffold learning (Mitrovic and Weerasinghe 2009).

Currently, support for STPA analysts is primarily provided through a handbook by N. Leveson and Thomas (2018), although regular workshops are also held. Whilst the handbook contains a wealth of information, it is not interactive or adaptive to the current user. Many learning institutions have textbooks, yet the teacher will still intervene with additional support as part of their pedagogy, providing personalisation for each learner, including through scaffolding.

2.1.4 Software Support for System Safety Analysis

Inherent in any software for doing an STPA analysis is some level of support that emerges from the user interface, which lays out the steps, terms, provides diagramming tools, and hyperlinks between related terms. Tools that offer this generic, nonadaptive support include: STAMP Workbench[1], SafetyHat[2], and XSTAMP[3].

Additional, intelligent support requires some kind of reasoning over the model to adapt to the user's particular analysis. In this category there are three tools for STPA.

SpecTRM, A State-Based System Model

SpecTRM is the first formal model used for STPA analysis. It is based on the formalisation of the definition of a hazardous control action and makes use of SpecTRM-RL, which uses a state-based representation of the system (Thomas 2013).

The formal model can be used to automatically generate a list of potential unsafe control actions described by the tuple: (SC,T,CA,Co), defined as:

- **SC**: Source controller, provided of control actions
- **T**: Type of control action (*Provided* or *Not Provided*)
- **CA**: Control Action, capable of being provided by SC
- **Co**: Context in which CA is provided or not

[1] Available at `https://www.ipa.go.jp/sec/stamp_wb/manual/index.html`
[2] User guide available at `https://rosap.ntl.bts.gov/view/dot/12034`
[3] Available at `https://github.com/SE-Stuttgart/XSTAMPP`

The context (Co) is described by a set of "context variables" associated with "context values". One example given (Thomas 2013, p.115) is for "train motion" as a context variable with the possible context values of "stopped" or "moving". The generation of potentially hazardous control actions is essentially an enumeration and Cartesian product of all possible values for each element in the tuple. Nine rules are defined that ensure the validity of the model (Thomas 2013, p.116), however the model is not capable of distinguishing the ninth rule:

> 9. To qualify as a hazardous control action, the action (SC,T,CA,Co) must be able to cause a hazard $H \in \mathbf{H}$, where $\mathbf{H}$ is the set of system level hazards. (Thomas 2013, p.116)

Thomas (2013) argues that the reduction of the set of automatically generated tuples to those that are hazardous is a manual operation, which supports the early stages of system development where the formal system model may be lacking. However, if it were possible to automate the reduction of the generated hazardous control actions (or only generate those reasoned to be hazardous), this could be used to provide support for the analyst by providing an explanation of why the control action may or may not be hazardous. Furthermore it provides a means of mitigating mistakes by calling attention to those control actions that can be reasoned to be miscategorised.

From the set of hazardous control actions described by the tuples, requirements defining behaviour to be avoided can also be generated.

WebSTAMP

WebSTAMP is a web-stack based tool for STPA analysis (Souza et al. 2019). In terms of intelligent support provided it also also uses the automatic generation of potentially hazardous control actions that were described by Thomas (2013). In addition it applies the rule-based approach of Gurgel et al. (2015) to include whether or not the software has identified a control action as hazardous to aid the user when they are manually reviewing the list.

The rules are manually created by the user to set certain parameters for the values that system qualities can take as always being hazardous. The example from Gurgel et al. (2015) is that if a train is moving and the doors are not fully closed, that is a hazard. Thus the rule allows the system to determine that the system states where the train is moving and the doors are partially or fully open are hazardous.

RMStudio

RM Studio have been developing an STPA module for their risk-management suite of software[4] (Björnsdóttir and Rejzek 2017). The intelligent support includes analysis of Unsafe Control Actions and Loss Scenarios using data from the control structure and progress checking of the analysis process. The progress check informs the analyst about all the defined elements and how they are used or not used. This can include issues such as no justification been documented, or an Unsafe Control Action not yet being linked to a Hazard. As proprietary software the underlying mechanisms are undisclosed.

2.2 Ontology and System-Safety

The techniques employed in safety analysis can be divided into those used prior to a loss (STPA) and the those used after (CAST). Within the realist framework, they can also be divided into those applied on a universal, class or a particular (Arp et al. 2015), as shown in **Figure 2.3**. Before a loss the analysis is conducted on a universal, which informs the design, manufacture, use and maintenance of a particular. The particular is monitored through provenance data. After a loss an investigation conducted on a particular is used to consider safety improvements on some universals and classes. These improvements are then applied on particulars of these universals or classes. **Example 2.1**, based on an accident report available at Skybrary (*Aviation Investigation Report: Loss of Rudder in Flight* 2007), highlights these distinctions.

Example 2.1

When the designers of the Airbus A310-308 conducted Fault Tree Analysis (FTA) to determine how it may fail, they were considering the universal A310-308 prior to any accident or incident. When the A310-308, with serial number (MSN) 597 underwent pre-flight checks on 6th March 2005, the crew were checking a particular prior to any accident or incident.

When (MSN) 597 landed later that day with its rudder missing (*Aviation Investigation Report: Loss of Rudder in Flight* 2007), it was a particular that was investigated after an accident or incident. When the investigators subsequently recommended an inspection program they did so for the class of aircraft with part number A554715000 series rudders (*Aviation Investigation Report: Loss of Rudder in Flight* 2007), again this was after the accident.

[4]Documentation (incomplete at time of writing) available at: `https://support.riskmanagementstudio.com/?section=stpa`

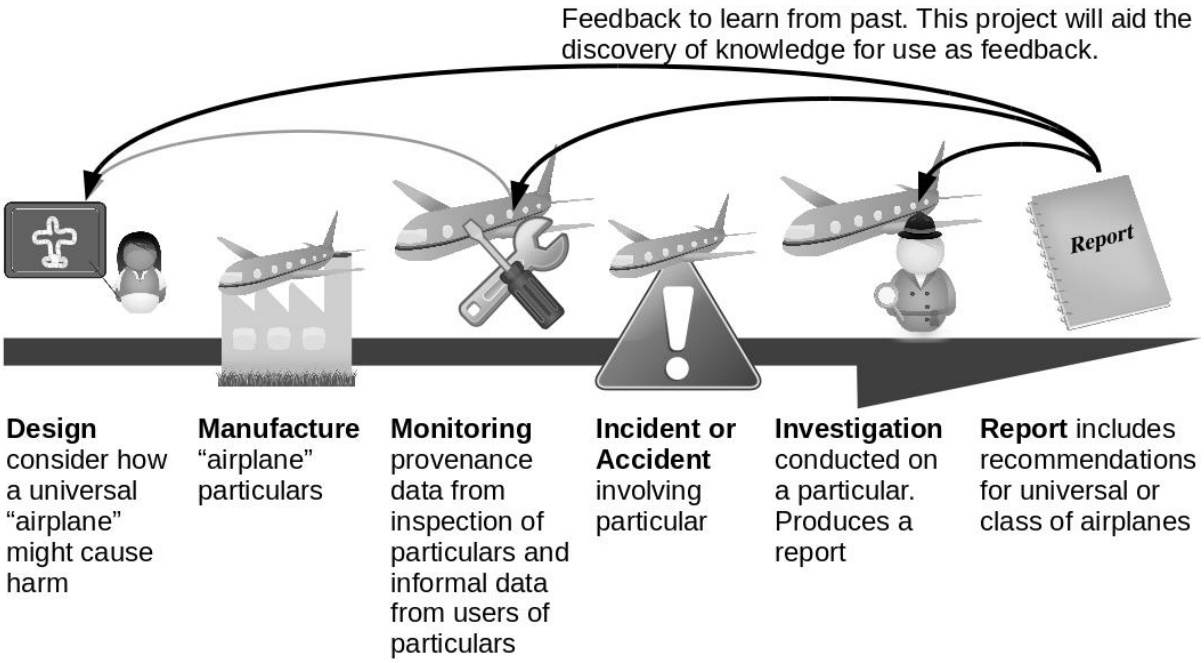

Figure 2.3: Illustrating the need for ontology in System Safety. Shows distinction between whether a universal, class or particular is being considered and highlighting how recommendations included in the report can inform system safety processes at each stage. An "airplane" is used to represent any system.

In STPA, STAMP can be conducted for a system that does not yet exist, considering the abstract kinds of events and kinds of components, i.e. universals. The intended use of this knowledge-representation is within a knowledge-based application to support analysis with STAMP. Therefore, it needs to be capable of describing hypothetical control systems and their behaviour including projecting the consequences of that behaviour, at various levels of abstraction. This is within the purpose of ontology as argued by Gruber (1995): as a conceptualisation of some simplified world-view for a purpose.

The key activities of an STPA analyst center around identification, such as identifying situations to be avoided, and communicating their findings, such as constraints that must be placed upon system behaviour. The formal logic with which ontologies are represented enable assistance in identification, as per the enumerative methods using the set-theoretic representation (Gurgel et al. 2015; Thomas 2013), however the additional information captured can do so more efficiently and for more terms.

Additionally, incidents caused by miscommunication have occurred, such as those surrounding units of measurement in the Mars Climate Orbiter incident (Stephenson 1999). STAMP recognizes this need for people and teams to share a common mental model of the system (N.

Leveson 2017, p.96). Ontologies not only ensure a common vocabulary (Noy and McGuinness 2001) but also provide a reference model, thus reducing ambiguity in communication. They're also machine interpretable (Noy and McGuinness 2001), which will enable additional intelligent tool support.

2.2.1 System Safety Related Ontologies

Aviation Safety Ontology

An ontology for aviation safety, aligned with Unified Foundational Ontology (UFO)[5] is available through INBAS (**IN**dicator **BA**sed **S**afety project) (INBAS 2018). The ontology is divided into sections for UFO, documentation, core, aerodrome, airline, air traffic management, maintenance and aviation (Kostov et al. 2017). It contains over 1700 classes and over 2000 individuals (INBAS 2018). This ontology was analysed for anti-patterns, which whilst syntactically correct do not produce the desired model (Ahmad and Křemen 2016). Anti-patterns were found in the parts corresponding to both UFO-A and UFO-B (Ahmad and Křemen 2016).

The Aviation-Safety Ontology describes both the systems and contains rules for safety events, an example is shown in **Equation 2.1** (INBAS 2018).

$$\begin{aligned} &\text{aviation-safety:Bird strike} \sqsubseteq \text{aviation-safety:Wildlife strike} \sqcap \\ &\exists \text{aviation-safety:has coliding [sic] object.aviation-safety:Bird} \end{aligned} \tag{2.1}$$

Although much of the ontology is domain specific it includes a Safety ontology aligned to a conceptualisation of Risk Management. It contains terms to reason about the findings of audits with regards to incidents and relate them to similar incidents (Kostov et al. 2017). This module is still tailored to the working procedure in the Aviation domain, with the issue of incidence reports, and does not include equivalents to terms essential to STAMP such as "Hazard", "Loss", or "Controller". STAMP is a non-domain specific model and a non-domain specific ontology is required.

[5] Discussed in the upcoming **Section 2.2.2**

Hazard Ontology

The Hazard Ontology (Zhou, Hänninen, Lundqvist, and Provenzano 2017) is designed to capture the definitions of hazards identified by Preliminary Hazard Analysis, which can then be used to further refine the definitions of these identified hazards (Zhou, Hänninen, and Lundqvist 2017) and identify causes of hazards (Zhou, Hänninen, and Lundqvist 2017). It provides an ontological foundation for describing hazards ground in Unified Foundational Ontology (UFO). It considers hazards as a kind of situation that can be brought about by events, and provides terms to capture additional information such as exposure, hazard elements, initiating roles and initiating factors.

The Hazard Ontology claims differentiation from STPA in that it formalizes the analysts' expert knowledge of the system (Zhou, Hänninen, and Lundqvist 2017), despite the set-theoretic model for STPA (Thomas 2013), and that it enables the exploration of the causes of hazards (Zhou, Hänninen, and Lundqvist 2017), despite step 4 of STPA: Identify Loss Scenarios (N. Leveson and Thomas 2018). Although the Hazard Ontology is designed for the results of Preliminary Hazard Analysis, the shared terms of situation-hazards, events, and the controller role suggest that it may be possible to extract some shared higher ontology module positioned between UFO and both the Hazard Ontology and the STAMP Ontology.

There are some fundamental differences in conceptualisation though. In Hazard Ontology an Initiating Condition or Hazard can be related to an Initiating Event with a 'trigger' relation, and Initiating Events can 'cause' Initiating Events. This conceptualisation caters to what Reiter (2001) calls *natural actions*, which are those that cause change in the system due to obedience to natural laws such as the Newtonian equations of motion. Thus if a condition exists where a piano is up in the air unsupported, that will trigger the "falling" event, which will cause the "piano crashing down" event. The "unsupported piano in the air" situation itself does not cause the piano to fall, that makes it possible for the piano to fall. It's the law of gravity that effects the falling, and ensures that the piano will always fall if it is possible that it can fall.

In STAMP the concern is with Control Actions, which are those events which are chosen to be actuated by some controller, as only these can be manipulated to prevent Hazards from occurring. The natural events are abstracted out of the model. So in STAMP the more appropriate definition would be to relate a 'Control Action' to a 'Situation' with a relation that denotes some 'Controller', who is capable of doing that 'Control Action' could decide to attempt to actuate

that 'Control Action' in that situation, i.e. The Baker House students have the capability to drop a piano off the roof, which they realize in a situation where the piano is on the roof, it's annual piano drop day, and they choose to drop the piano. To remove agency from the controller and state that the situation causes the event is to commit the fallacy of reification; the situation has no agency to effect change. So any bridge between Hazard Ontology and a STAMP Ontology will need to account for the differences between UFO Events and STAMP Control Actions while avoiding this fallacy.

Hazard Causes and Consequences

Another ontology supporting hazard identification is proffered by Vargas and Bloomfield (2015). They also define hazards as a situation, but also define "Cause" and "Consequence" as situations with no relation to events or actions that transition between situations. They do include a Process definition, which is used to describe system behaviour and can be defined as the behavioural location and scope of a hazard. Thus this definition lacks the terms to reason about how and why a "Cause" situation could transition to a "Hazard" situation, and how the "Hazard" could transition to a "Consequence" situation. Given situation's exist and do not change in and of themselves, it seems odd to say a situation is a cause, rather than it being one in which it is possible to transition to a hazard, and the event transitioning to the hazard then is the causal part.

The Vargas and Bloomfield (2015) ontology also contains a mereology of systems that includes terms for "System", "Subsystem", and "Part", which are all kinds of "Component". They've understood that "Systems" are a recursive structure, until the fundamental components that can no-longer be split into constituent parts are reached. The use of the additional "Subsystem", "Part", and "Component" terms is an attempt at scoping hazards, however they enforce the representation of a system being modeled to the level of granularity chosen. Should an analyst wish to decompose some "Part" in the ontology to examine its constituents, they're unable to do so without redefining it as a "Subsystem", which suggests an ontological error. Even in the case where the parts were the elementary particles (such as quarks), there are multiple theories reasoning hypothetically about the composition of these (such as preons), and ontology should be capable of representing the hypothetical in order to support reasoning and creative thought.

STPA-Sec Ontology

An ontology for STPA-Sec, which is the application of STPA to Security is defined by Pereira et al. (2019). The ontology reuses the "Hazard" term from the Hazard Ontology, and the "Hazard triggers UnacceptableLoss" relationship already discussed. These are combined with terms from security ontologies and terms from STPA-Sec (the methodology, not a pre-existing ontology). Given its different purpose much of the security related terms are superfluous to a plain STPA analysis, however the inclusion of these existing ontologies effects the definition of STPA terms that they subsume. Therefore rather than defining controllers, actuators, and sensors in terms of the control-systems theoretical foundation of STAMP, they're defined as "STPASecElements" (where "Controller" is subsumed by "ModelledEntity", which is subsumed by "STPASecElements"), which are a kind of "Asset", where an "Asset" is an inherited term taken from ontologies that defines it in terms of aircraft and another that defines it in terms of items of value to organisational objectives (Pereira et al. 2019). From these it's understood that an asset can be a "function, security measure, human, or information, [and] function is realized as part of the *Controller* entity" (Pereira et al. 2019, p.306).

Control Systems theory is central to STAMP, N. Leveson (2017, ch.3) dedicates a foundational chapter to explaining how it is understood in STAMP. The definitions in the STPA-Sec ontology fails to capture how these "STPASecElements" are understood in STAMP, and instead differentiates them taxonomically by their use in the STAMP framework, typified by "ModelledEntity". This is an ontological flaw, it's describing how the terms are used in the artifacts produced in the analysis, rather than the nature of their being: their ontological nature.

Summary of Safety Ontologies

The authors of the ontologies discussed are motivated by the capability for ontology to aid in the improvement of system-safety analysis. Notably Pereira et al. (2019), who authored the STPA-Sec Ontology, found in their evaluation it helped analysts to identify more security scenarios than when not using it. There are however ontological modeling issues and refinement of the representation of STAMP terms to be addressed.

2.2.2 Modular Ontologies

The STAMP ontology uses terms that are subsumed by more generic terms from more general theories. As STAMP is an ill-defined domain, these more general terms are modularised *a priori* to aid in managing the complexity of the model and the dynamicity of the module edited by the analyst (Thakker et al. 2011). The methodology of Thakker et al. (2011) uses three levels to organise the modules:

1. **Upper Ontology**: to cover base concepts from a chosen theoretical framework
2. **High-level reusable domain layer**: connects the Upper Ontology to the Case specific layer by defining domain concepts in terms of the upper ontology at a granularity such that it can be used with multiple specific cases.
3. **Case specific layer**: defines concepts to the specific use case.

For the STAMP ontology, a case is taken as a distinct system-safety analysis model. Therefore this is left to the analyst to define for their system by extending the terms in the higher level modules. The Upper Ontology layer is required to define the most abstract concepts to aid in defining the High-level reusable domain layer.

Between the two layers needs to be all the definitions to capture the STAMP model. This includes STAMP specific terms as well as terms from its conceptualisation of the underlying control systems theory. The control systems layer needs to be capable of capturing parts of the system, such that it defines actuators, sensors, and controllers, but the structure of the system is of little concern in STPA and needn't be captured. However, the behaviour of the system is a primary concern for system safety analysis, and so a suitable representation is required.

Top Ontologies

These are the top-level, most abstract ontologies that are used for organising and guiding the precise definition of the lower modules.

Basic Formal Ontology (BFO) BFO (Arp et al. 2015) is a top-level ontology used by the domain ontologies that comprise the Open Biomedical Ontologies (OBO) Consortium, among others. BFO's established use in the biomedical domain suggests it's successful at modeling complex and critical systems. It commits to a philosophy of realism, meaning it commits to only representing reality and not the linguistic, conceptual or theoretical.

The implication of a commitment to realism for a STAMP ontology means that any linguistic, conceptual, or theoretical thing would have to be defined as such. The very concept of "system" is one such term. In realism there is no entity that is a system. Instead the entity, such as a kettle, exists as an object: a material entity. The concept of system in realism is a kind of "Concept", and so if one wished to examine the kettle using a systems model, it would be necessary to define "system" as some kind of theoretical modeling conceptualisation, and relate objects such as "kettle" to the "system model of kettle". The same applies to many terms in STAMP, and the STPA analysis would then be conducted on these system model conceptualisations.

The trade-off between truth and practicality needs to be considered with regards to a commitment to realism. The realist representation is truthful, and so is more accurate. It's also more versatile as a convenient lie of omission for one purpose can prevent re-application to another purpose. However for an STPA analyst to use it would require training and a great divergence from how they usually conduct an STPA analysis, unless it could be abstracted away in the tools they use.

The core of BFO distinguishes between a Continuant, which may be independent or dependent, and an Occurrent. For a kettle example:

- the heating element is an independent continuant
- the heating function is a dependent continuant, it's dependent on the heating element, power and water being present.
- the heating functioning is an occurrent, this is the heating element performing its heating function and is a process.

BFO also includes useful definitions to describe roles, objects, regions, sites, and fiat boundaries. It also considers dependency at its core and distinguishes between a disposition for an event and the occurrence of that event, which may be useful. Del Frate (2014) has mapped failure onto the terms continuant and occurrent. There is an effort to create a top-level ontology for systems engineering using BFO (Jenkins 2018), however it's in the early stages of development.

Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE) DOLCE is a top-ontology with a cognitive-bias (Masolo et al. 2002), which contrasts it against BFO's commitment to realism. It's designed to capture ontological categories underlying language and human commonsense (Masolo et al. 2002). DOLCE should be able to capture terms like "system",

and "situation" without needing to first define what a conceptual thing is: its commitment is to capturing cognitive artifacts. It includes terms related to mereology, participation, qualities and functions. However, it's not intended to be a "universal" standard ontology (Masolo et al. 2002), and was one of the foundational ontologies of UFO (Guizzardi, Botti Benevides, et al. 2021).

General Formal Ontology (GFO) GFO (Herre 2010) was envisioned to be a unifying foundational ontology. It's structured with layered modules, which include definitions for objects and processes. It was designed for use in applications in the socio-technical systems domain, particularly medical, economical, and sociological.

GFO also has a commitment to realism that differs from that of BFO. In BFO the ontology must be based on universals in reality, instead of concepts. GFO argues that there is no representation of reality without concepts (Herre 2010). Therefore in GFO the concept of "System" can be defined as a term without first defining "Concept" to subsume it, as the concept of "System" exists in society.

GFO was used as one of the foundational ontologies of UFO (Guizzardi, Botti Benevides, et al. 2021).

Unified Formal Ontology (UFO) UFO (Guizzardi, Botti Benevides, et al. 2021) is a multi-part upper-level ontology, development on which is ongoing, therefore an "Essential" UFO is also used as a reference version (Guizzardi and Wagner 2011), which is indicated by the acronym "eUFO". UFO, and thus in deference eUFO, are modular, including a base layer (0) defining terms like "Thing", "Individual" and "Universal". This is extended to an A-part and B-part, where A is concerned with material things, such as matter and qualities, whereas eUFO-B, shown in **Fig. 2.4**, describes events.

UFO was motivated by the desire to provide ontological foundations for conceptual modeling. The initial UFO attempted to unify DOLCE (Masolo et al. 2002) with General Formal Ontology (Herre 2010), and has since continued to unify fundamental conceptual modeling notions including types, mereology, particulars, roles, and the aforementioned events (Guizzardi, Botti Benevides, et al. 2021). It aims to be a means for describing reality and its accounting for in human cognition.

The practicality of using UFO as a top-ontology for system safety is attested by its use in the

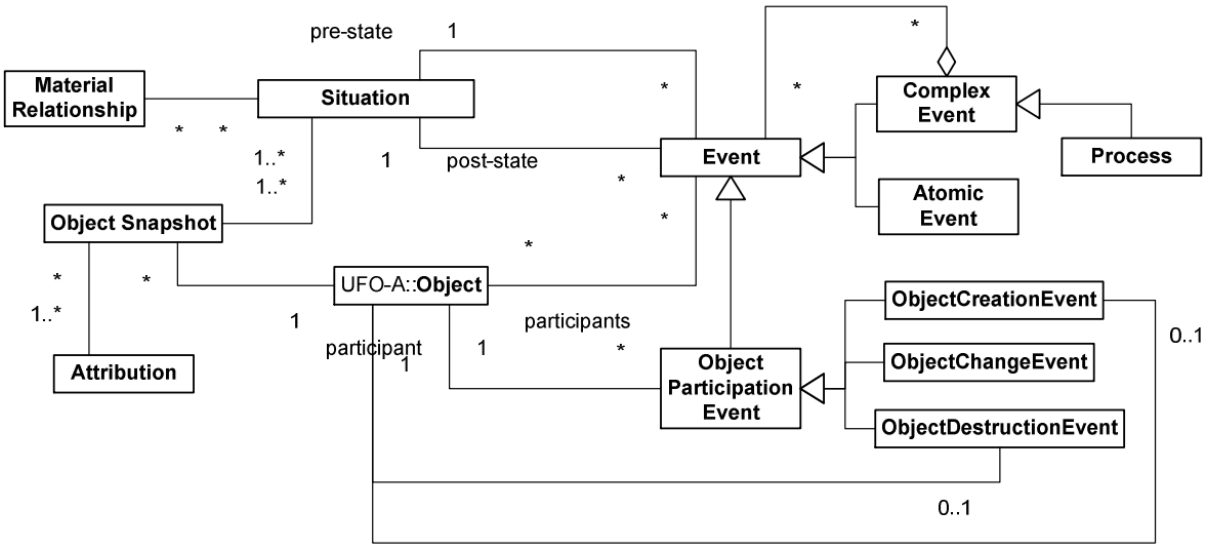

Figure 2.4: Depiction of the eUFO-B ontology (Guizzardi and Wagner 2011).

Aviation Safety Ontology, Hazard Ontology, and thus STPA-Sec Ontology, discussed previously.

Structural Ontologies and Formal Representations

One aspect of control systems theory utilized by STAMP is the different kinds of systems and their interactions. Although there is little need for a "has part" relation, it would be wise to consider an ontology that accommodates a mereological definition such that any ontology created during an STPA analysis can be re-used in other tasks.

Semantic Sensor Network The Semantic Sensor Network (SSN) (Haller et al. 2017) family of ontologies defines many of the required system terms making it a suitable candidate for extension. However the purpose differs enough that reusing terms could be a source of confusion: an Actuation in SSN is the instantiation of some kind of occurrent that is recorded in the ontology, whereas in STAMP it is the capability that inheres in some kind of actuator that can be realised in an occurrence. For example, SSN would record that WM30 is a wind-speed sensor with its operating range, and that at 10:03 an instance of a WM30 sensor recorded a wind speed of 5ms (Group 2005). Whereas a STPA would record that there is some kind of wind sensor that has the capability to feedback wind speed data.

Part-Of Relations Mereology is the study of parts and their sums, which is the wholes they form. Given system-theory is a decompositional and compositional theoretical model, mereology provides a calculus for reasoning with the parts of a system. Here it is defined formally using

the same notation as Simons (1987).

x is a part of y: $x < y$

Reflexivity: $y < y$

Antisymmetry: $x < y \wedge y < x \rightarrow x = y$

Transitivity: $x < y \wedge y < z \rightarrow x < z$

x is a proper part of y: $x \ll y \doteq x < y \wedge \neg y < x$

x overlaps y: $x \circ y \doteq \exists z(z < x \wedge z < y)$

x is disjoint with y: $x \wr y \doteq \neg(x \circ y)$

These definitions don't account for different types of whole/part relations, just that "x is part of y" without considering what "part of" means. Different taxonomies of these relations have been proposed, which include component-integral relationships (Odell 1994; Winston et al. 1987). These are discussed as parts of the whole that have an additional functional relation to the whole, which distinguishes them from pieces of a whole (Odell 1994; Winston et al. 1987). This can be formally defined following the same definition used by Vieu, who in turn uses a definition from Casati and Varzi (Vieu 2006).

Functional part: $x <_f y \doteq x < y \wedge \phi(x, y)$

Vieu (2006) has no need for antisymmetry in the definition. In the definition ϕ is used to denote some relationship between x and y that is understood to be functional.

This is perhaps the simplest definition of part-hood that is suitable to describe a system structure. To make the term more in-keeping with systems language, it can be denoted as componentOf(x, y) or hasComponent(y, x) The underlying idea can be extended in a number of ways as the representation requires, for example, this rule can be used to define distinct entities that interact in the same environment.

$$\text{interact}(x, y) \doteq \exists e(x \ll e \wedge y \ll e \wedge x \wr y \wedge \psi(x, y))$$

Behavioural Ontologies and Formal Representations

The second aspect of control systems theory utilized by STAMP is that they behave. Behaviour is understood herein as a sequence of actions that when realized occur in time but that can also be conceptualised for analysis and reasoning.

Behaviours And Functions Of Technical Artifacts This formal definition of technical artifact behaviour is based upon the definitions of perdurant found in DOLCE[6] and also makes use of mereological terms (Borgo et al. 2009). It is intended for use with technical artifacts, of which socio-technical systems may be considered a subset.

A ternary relation, read as "b is the behavior of the technical artifact a in event e" (Borgo et al. 2009, p.11) is introduced as a primitive:

$$Beh(a, e, b)$$

Borgo et al. (2009) then consider possible behaviours, firstly by restricting what is logically possible to occur. Generalised engineering perdurants (GEPD) excludes those which are not logically possible. Physically possible perdurants (EPD) are those which are physically possible, and actual possible perdurants (APD) are those which are actually possible. Parts of actually possible and physically possible perdurants are actually or physically possible respectively. Actual possibility holds for the sum of actually possible perdurants.

[6] Equivalent to occurents, some kind of thing that "extend[s] in time by accumulating different temporal parts, so that, at any time they are present, they are only *partially* present, in the sense that some of their proper temporal parts [..] may be not present." (Masolo et al. 2003, p.15) A typical example is in doing some process, where at any point during the process the past-parts of it are no longer present and future parts are not yet.

$$APD \subseteq EPD \subseteq GEPD \subseteq Perdurants$$

$$APD(e) \wedge Part(e', e) \rightarrow APD(e')$$
$$APD(e) \wedge APD(e') \rightarrow APD(e + e')$$
$$EPD(e) \wedge Part(e', e) \rightarrow EPD(e')$$

Axioms for possible and impossible behaviour are then defined.

$$PossBeh(a, e, b) \triangleq Beh(a, e, b) \wedge EPD(e)$$
$$ImBeh(a, e, b) \triangleq Beh(a, e, b) \wedge GEPD(e) \wedge \neg EPD(e)$$

Following this, relations for input and output perdurants are defined, which could be used to represent the actions of a controller as an agent. More axioms are provided to represent conditions, behavioural constraints, causation and interaction with the environment.

The distinction between kinds of possibility with regards to behaviour goes beyond the requirements for STAMP, but could be useful when considering the rule-based approach of Gurgel et al. (2015) and supporting an analyst by identifying which control actions are potentially hazardous. To do so would require a manner in which to identify the kind of possibility from reasoning rather than assertion.

Situation Calculus Situation calculus was introduced as a formal language that could be used by a computer program to decide what to do in order to reach a goal via a strategy (McCarthy and Hayes 1969). It is based on representing the world in terms of discrete interacting automata (McCarthy and Hayes 1969), which has sufficient similarity to the systems model to be acceptable, and follows the perdurant representation of time through fluents, where a fluent[7] is a predicate that includes a situation (Reiter 2001), i.e. they are facts that are only true in certain situations. It was designed with intention of answering questions, which also arise in

[7]Strictly a relational fluent; functional fluents are not used in this thesis.

safety analysis:

1. What will happen next in a certain aspect of the situation?
2. What will happen if I do a certain action?
5. Can I figure out how to do this or must I get information from someone else or something else?

(From McCarthy and Hayes (McCarthy and Hayes 1969, p. 4))

Situation calculus has many compatible features, including definitions of situations and fluents, which have been discussed prior to introducing it here. Also it has a manner of determining causality; action dependencies; action effects; strategies, which are a combination of actions; and knowledge, which might for the basis of a representation for beliefs (McCarthy and Hayes 1969). It is typically associated with planning tasks in AI, where a plan is found between a given state and a goal state (Reiter 2001). This planning task bears strong similarities with the kind of reasoning required for safety and reliability analysis.

However, the situation calculus is not without problems, most notably the Frame Problem, which has several proposed solutions (McCarthy and Hayes 1969; Shanahan 1997; Reiter 2001; Mueller 2015). Among these solutions is the Event Calculus, which provides solutions to many problems the original Situation Calculus does not (Shanahan 1997; Mueller 2015), however it is also based upon linear time, whereas Situation Calculus uses branching time (Mueller 2015). For the purposes of Safety Analysis, branching time is a natural approach, allowing the analyst to consider multiple possible futures from a given situation. An effort has been made to unify the two calculi, however the computability of the unified calculus is unknown (Bennett and Galton 2004).

Within Situation Calculus, a "simple solution" to the Frame Problem is offered by Reiter, which has been used and tested in the Toronto University robotics laboratory (Reiter 2001). The solution provided is limited to deterministic, primitive actions (Reiter 2001). However, Reiter also offers Golog, which can overcome these difficulties.

Golog is a logic programming language for an extended situation calculus with a Prolog interpreter that can be used for applications in dynamic domains, including robotics, planning and data applications (Reiter 2001). Variations of the interpreter offer extensions to manage tempo-

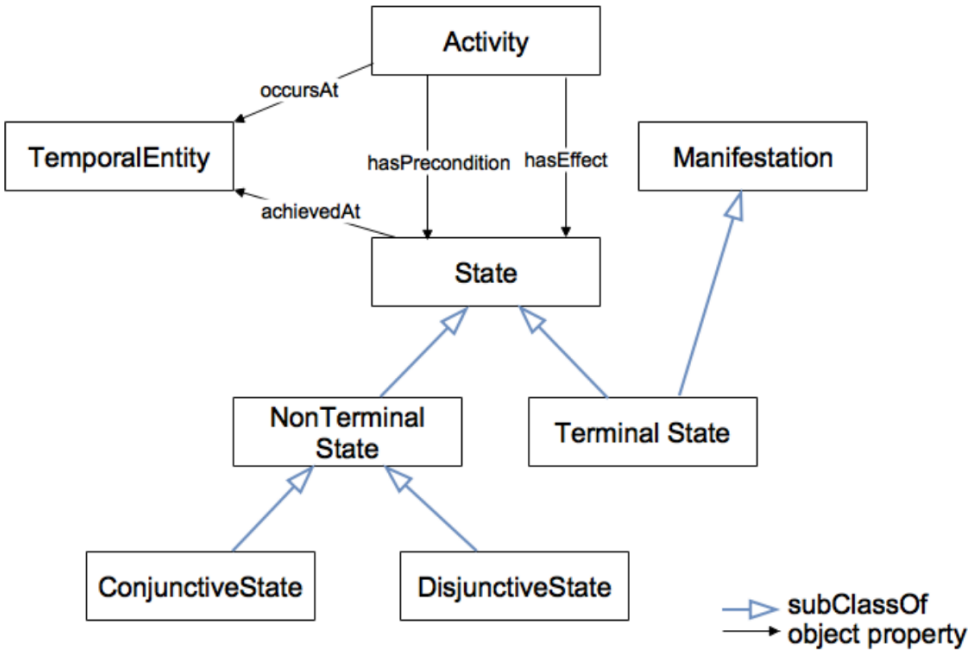

Figure 2.5: Depiction of the Activity Specification ODP (Katsumi and Fox 2017).

ral information, exogenous actions, interrupts, environmental sensing, knowledge, and stochastic actions. This suggests it could be an ideal tool to use in safety analysis, capable of application offline, as well as for a system in use.

Activity Specification Ontology Design Pattern (ODP) The Activity Specification ODP (Katsumi and Fox 2017) is based upon the notion of causality found in Event Calculus, which is related to Situation Calculus (Kowalski and Sergot 1986). Situation Calculus makes use of a representation of reality "as a system of interacting discrete automata" (McCarthy and Hayes 1969, pg.9). The term 'state' used is equivalent to 'situation', 'activity' is equivalent to 'event'.

This representation assumes an activity has a precondition described by a state, and that activity can produce some result that is a new state. This implies that an activity *can* occur when its preconditions are met. This design pattern could be used to describe the process leading to a loss event occurring.

Katsumi and Fox (Katsumi and Fox 2017) suggest the pattern can be adapted to include other ontologies for activities, temporal entities and states, as well as be expanded to include participants. In comparison to Situation Calculus as defined by Reiter (2001), it lacks the relations to describe the states (which are expected to be included from another ontology module), and therefore lacks the relations to describe changes in states brought about by activity.

UFO-B UFO-B is one part of the UFO top ontology already discussed, it's the part that defines perdurants (those things that occur) (Guizzardi, Botti Benevides, et al. 2021), depicted in **Figure 2.4**. It includes terms to aggregate atomic events into complex events, participants, and situations. It also contains the same pattern of relating events to situations as the Activity Specification ODP (Katsumi and Fox 2017), however under UFO-B the pre-state is simply what the situation was prior with no commitment to it being a condition of the event or activity. In comparison to Situation Calculus as defined by Reiter (2001), it also lacks the relations to describe changes in situations brought about by events.

2.3 Contingent Scaffolding

Contingent Scaffolding is presented by D. Wood, Bruner, et al. (1976) as a process enabling the learner to accomplish a task beyond their current capabilities, which is one key goal of supporting the non-expert analyst. This is achieved by some more knowledgeable other controlling the elements of the task that the learner cannot yet accomplish, allowing them to concentrate their efforts upon those that they can (Daniels 2010). Day and Cordón (1993) demonstrated scaffolding required fewer examples and explanations, as well as providing better maintenance of learning.

The term "scaffolding" was coined to describe how mothers were able to teach children as young as three to complete a task that should not be possible for pre-operational children, as per Piaget's theory, to accomplish. Wood and Wood (D. Wood 1998; D. Wood and H. Wood 1996a; D. Wood and H. Wood 1996b) distilled two main observations from the studies for successful scaffolding: help was offered as soon as the child struggled, and help was incrementally withdrawn when the child succeeded. They later expounded the principals of "contingent scaffolding" (H. Wood and D. Wood 1999) as:

- Help is provided expeditiously when the learner is in trouble
- Help is increased as the learner requires, until the solution is reached
- As the learner succeeds, support is withdrawn

The term "contingent instruction" was coined by the same group to describe this scaffolding method of tutoring where the child has access to help when struggling but is not held back by direct and intrusive teaching (D. Wood 1998). Inherent within contingent instruction is the

offering of support in structuring the task at hand, which also reduces uncertainty. It's argued that uncertainty in the task at hand distinguishes the learner from the expert and can make learning more difficult (Daniels 2010). Daniels (2010, p.109) suggests that the support offered be within the child's Zone of Proximal Development, as per Vygotsky's theory, and suggests six levels of control:

- Level 0: no assistance;
- Level 1: a general verbal prompt ('What might you do here?');
- Level 2: specific verbal ('You might use your computer tools here');
- Level 3: indicates materials ('Why not use a graph plotter?');
- Level 4: prepares materials (selects and sets up tool);
- Level 5: demonstrates use.

The learner's behaviour is observed to determine whether intervention is required, the tutor then moves through the levels of support. What it means to be in "trouble" isn't defined, but is left to the tutor to intuit. The number of levels vary, between 4 and 5 in QUADRATIC (H. Wood and D. Wood 1999) or the 6 above (Daniels 2010); the only guidance being that they should increase in depth or interference until physical intervention is undertaken.

Scaffolding has been successfully applied in Intelligent Tutoring Systems (ITSs), typified by providing one-on-one micro-tuition with graded support for multi-step problem solving in formalised domains (Du Boulay and Luckin 2001). One notable example for the safety domain is Sherlock II, an apprenticeship-style learning system providing a coached practice environment for troubleshooting complex electronic circuits in the domain of aviation. The amount of help provided by the system when the analyst requested it was based upon their prior requests for help and student model, becoming increasingly directive (Katz and Lesgold 1993).

D. Wood and H. Wood (1996b; 1998) draw the similarity between Anderson's Production Rule approach developed from ACT theories (Anderson 1993) and contingent instruction based on the *condition-action* pairs it results in. Anderson's production rules worked with defined goals and sub-goals (Anderson 1993), in well understood domains: The Towers of Hanoi, LISP, and Geometry (Anderson 1993). However, D. Wood and H. Wood (1996a; 1996b) also expressed concern with the lack of flexibility and inability to handle novelty. AI education tools that

have succeeded Anderson's work have addressed how the tool is integrated into the classroom, collaboration, as well as selection of tasks from within the Zone of Proximal Development and selection of language used when communicating with the learner (Du Boulay and Luckin 2001).

The concern in implementations regarding a lack of flexibility spans more than 2 decades (Du Boulay and Luckin 2001; D. Wood 2018; D. Wood and H. Wood 1996b), the difficulty arising from the capability of a learner to approach a problem in an unexpected but valid way. Previous work on ill-defined domains and tasks indicates various strategies have been successful, including constraints (Lynch et al. 2009) that can check if certain properties of a solution are present or not. The violation of some constraint indicates a need for intervention (Ohlsson 2015). Therefore the proposed contingency scaffolding framework outlined in **Chapter 4** uses constraints based on situational calculus and a domain ontology to provide scaffolding flexibility in the context of system safety analysis.

In order to support a STPA Analysis Support Framework providing Contingent Scaffolding, a history of the ontology authoring process is used such that the artificially intelligent tutor can determine if a learner has resolved an issue, or if a similar issue has arisen prior. With the typical synchronic manner of ontology authoring, only the ontology as defined at the time of viewing is available for viewing. Pertinent historical information is then captured in an extraneous user-model. The alternate proposed herein is to use Situational Calculus for ontology authoring to meet this requirement of a diachronic view of the ontology being authored, such that how the ontology was authored can be considered.

2.4 Supporting Ontology Authoring

The motivation to provide support to non-expert authors drives the desire to capture as much information regarding their authoring process as available, within a formal framework to support in-depth querying.

2.4.1 Existing Tools

Ontology authoring support tools that analyse the static ontology, include OOPS! (Poveda-Villalón et al. 2014), OntoDebug (Schekotihin et al. 2018), and BOADiS (Denaux 2013). These three exemplify different approaches to supporting authors. All three only have a synchronic view of the ontology, and all three are run repeatedly as the ontology is being authored.

OOPS! is an "OntOlogy Pitfall Scanner!" designed to semi-automatically catch common pitfalls beginners make when authoring ontologies (Poveda-Villalón et al. 2014). It uses a catalogue of common mistakes that it can identify, should one occur the author is informed so that they can correct it and re-check with OOPS!

OntoDebug is an ontology debugging tool using author-defined test-cases (Schekotihin et al. 2018) similar to unit-testing in software authoring. A user can define statements that the reasoner should be able to prove are true or false. The tests can then be run on the defined ontology to check the test statements. Any statement's truthiness that can't be proved is reported with a view of the possibly faulty axioms to aid the author resolve the issues. The tests can be run as frequently as the author desires.

BOADiS (Denaux 2013) is built on top of the Perico dialogue framework for ontology authoring. BOADiS has dialogue actions to encourage/discourage edits, checking implications, and checking justifications. This enforces a tight feedback loop and makes the ontology as understood by the tool more transparent to the author.

These tools all provide useful feedback with synchronic views of the ontology being authored. A diachronic view including the history of authoring would aid OOPS! to tailor feedback for repeated pitfalls, OntoDebug to rank possibly faulty axioms by recency and inspect related retracted axioms to determine if undoing an edit could resolve an issue, and aid BOADiS in tailoring its checking dialogue actions to be taken more frequently on facts with more related retractions.

2.4.2 Capturing Authoring History

The reification of the asserted triple, as done with 'rdf:Statement' (Hayes and Patel-Schneider 2020), or use of a Singleton Property (Nguyen et al. 2014), permits capturing of additional information such as who authored the triple and at what time. This additional information added to the static view of the current ontology enables answering questions about who, when, and pace. Such questions can aid in finding team-members with expertise in particular knowledge domains, finding who made what edits, or when knowledge on a domain was updated.

With times related to triples the sequence of assertion can be determined, however, retractions are lost as the triple is no-longer present in the ontology. Therefore no analysis tool with a synchronic view of the ontology can tell if a triple has been asserted, retracted and asserted

again, which might indicate some uncertainty as to its truthfulness. Or if some retracted triple might resolve some pitfall in the ontology or cause some test-case to succeed. Such information for database systems can be captured in a database log, or via Event Sourcing, both of which can be formalised by Situation Calculus as described by Reiter (2001).

In Situation Calculus the database log is the sequence of actions that update the database (Reiter 2001), denoted as $do([a:1 \ldots a:n], s0)$ or abbreviated as $do(\mathtt{T}, s0)$. This log is the source of events from which current and past states can be determined plus, being Situation Calculus, it can be queried directly. Furthermore, this log can be expanded to include other pertinent actions the author or agents may undertake, such as those used in this project: consulting help materials and offering advisory interventions.

Situation Calculus actions can still reify the triple and capture author and time information as well as provide static views of the ontology for analysis via existing tools. Furthermore, Situation Calculus enriches the information available about the ontology authorship by:

- Recording retractions, and thus edits
- Querying some past situation (Reiter 2001, p.76)
- Querying over all past situations (Reiter 2001, p.74)
- Capturing author actions in addition to those which alter the ontology

By using this framework, not only can the ontology be queried in order to facilitate the authors, but also the process of authoring. Furthermore, additional information is captured, such as who asserted what, when and where the author is focused. If additional procedural knowledge can be captured regarding the authoring process, as is the case with STPA, then the authors' progress can be compared against this. Thus the framework supports queries to answer questions such as:

- Who asserted what?
- Who asserted in this subdomain, and thus might be knowledgeable about it?
- Has this fact always been true, i.e. existed in an imported top-ontology?
- Has an author retracted a top-ontology fact?
- Has some fact been asserted and retracted multiple times? Perhaps a sign of confusion or conflict.
- Was one assertion made prior to another? What was the time difference?

- When was this assertion made? Perhaps new information has arisen to challenge its validity.
- What will be the effect of asserting or retracting? (Without doing the assertion/retraction)
- Can an assertion or retraction be made?
- Where is the author in the process?
- Is the author backtracking?

A subset of the questions from this list are used to reflect upon the framework in **Section 4.2.1**.

2.5 Related Work Conclusions

System Safety Analysis is a valuable aspect of System Safety dependent upon the cognitive abilities of the analysts who undertake it. It's an ill-defined task that could benefit from scaffolding, which should improve the quality of the analyses conducted by non-experts. This scaffolding could be provided in an automated manner provided software and the analyst can reason with a shared model, for which ontology is ideal. Therefore this thesis is undertaken to explore the provision of Contingent Scaffolding using ontology to this valuable domain of System Safety Analysis.

STAMP is selected as a system-safety model for initial exploration of using ontology to support System Safety Analysis because the existing set-theoretic representation is already a formal model that can be used to inform the ontology. STPA is selected for the analysis process because it reasons with hypothetical systems, for which the requirements are broader than for recording past events as could be done with CAST.

When considering existing ontologies for re-use, there are unsatisfactory elements in the safety and systems ontologies discussed, mostly in lacking terms to reason with branching time, but also in system representation, and not adhering to how terms are defined in STAMP. Therefore these ontologies are used to inform, but a modular ontology will be authored.

The top-ontologies differing opinions of realism would require careful navigation as many STAMP terms are conceptual representations of some real thing, such as system and situation. As this thesis intends to assess the practicality of the use of ontology in this model rather than enter a philosophical debate, a loosely defined top-ontology will be used such that the philosophical-ontologist can re-align this work with their own understanding of realism.

There are few requirements for defining the mereology of the system in STAMP. The primary concern for parthood is with function rather than structure, and so the existing ideas of functional-parthood can be re-used in the STAMP ontology.

In STPA there is the requirement to reason about hypothetical futures, which imposes subscription to a theory of branching time. The behavioral ontologies discussed tend towards a similar pattern of pre-condition and post-situation in some event. However, these ontologies lack sufficient relations to provide for the simple solution to the frame problem as defined by Reiter (2001), which provides some means to reason about the future even if not completely general. Therefore it's proposed to take inspiration from this common pattern to inform a ontology module capable of capturing sufficient information that it can be used with a Situation Calculus reasoner.

Contingent Scaffolding has been demonstrated to provide benefits in ill-defined domains and through the use of constraints the lack of flexibility issue can be addressed. Additionally there is no standard for the levels of support to be offered, and so this must be defined for this use-case. In this thesis it's hypothesised that a diachronic view of the ontology authoring process can provide additional information that can be exploited when providing support. Therefore Situation Calculus is used again to define a framework for Contingent Scaffolding, such that the possibility of an action in a situation provides the flexibility and the log of actions is used to inform the level of intervention.

Chapter 3

Ontology for STAMP

This chapter defines an ontology for STAMP[1], derived from ontologies for control systems, and situations, which by necessity are also defined. The purpose of this ontology is to capture a STAMP model that results from STPA such that intelligent software support may be provided to the analyst as they extend this ontology with the case they are examining. This ontology is later integrated into such a software tool providing support to an analyst, including exploiting a reasoner to aid in identifying hazardous actions and to intervene when appropriate.

3.1 Background and Problem Definition

The ontology is described here using Description Logic. For compatibility with OWL-DL[2] the DL specification which corresponds to $\mathcal{SHOIN}(\mathcal{D})$ (Baader et al. 2017) is used[3]. The architecture is depicted in **Figure 3.1**. `Top`, and `Situation` comprise the "Upper ontology layer", they cover the theoretical frameworks that describe control systems and their behaviour via situations, which are foundational to STAMP. `Control System` and `STAMP` are the "High-level reusable domain layer", they make the connection between the theoretical "Upper ontology layer", and the "Case specific layer". The final "Case specific layer" is not defined as this is the task of the analyst using the system: to extend the provided ontology to describe their specific case. This

[1] Available with example applications at: Paul S. Brown (Apr. 2022). *PaulBrownMagic/STAMP-Ontology: Initial Release*. Version v1.0.0. DOI: 10.5281/zenodo.6489774. URL: https://doi.org/10.5281/zenodo.6489774

[2] A dialect of the Web Ontology Language (Lacy 2005)

[3] STAMP ontology only makes use of the $\mathcal{SHI}(\mathcal{D})$ subset of the popular OWL-DL. This is a DL with $\top$, $\bot$, conjunction, disjunction, negation, existential restriction, value restriction, transitive roles, role hierarchy, and inverse roles (Baader et al. 2017).

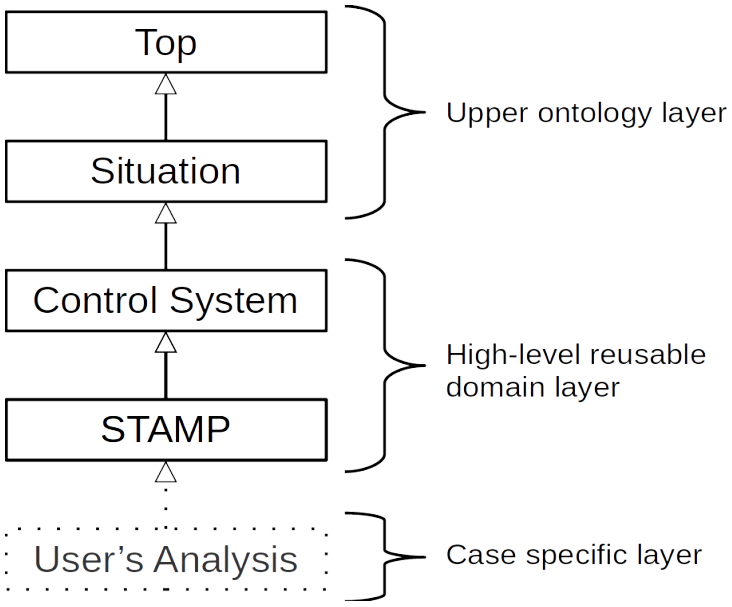

Figure 3.1: STAMP Ontology Architecture: modules depict ontologies with the arrow denoting a dependency on the definitions in the indicated module and above

architecture was chosen *a priori* to aid in managing complexity and dynamicity (Thakker et al. 2011) as discussed in **Section 2.2.2**.

The requirement imposed by STPA to consider systems that may not yet exist beyond a vague conceptualisation inhibits adherence to any top-ontology committed to realism, as understood in BFO as opposed to GFO[4]. There still remains multiple suitable top-ontologies, however for the purpose of independence, no adherence to any existing top ontology is claimed. Instead a tiny top ontology is defined with the required terms, which have counterparts in many top-ontologies, providing organisational structure to the subsuming terms.

The next "Upper ontology layer" module, the `Situation` ontology, fulfills the requirement to describe behaviour in terms of kinds of events and to consider the consequences of actions systems may take. Thus an adherence to a philosophy of branching-time is a prerequisite so that multiple futures can be considered.

The penultimate module and first one of the "High-level reusable domain layer" (Thakker et al. 2011), is required to describe control systems, a foundational theory for STAMP (N. Leveson 2017, ch.3). Control System theory is a high-level reusable domain, however, due to pragmatics the `Control System` ontology is offered with caveat. This `Control System` ontology is defined with regards to what is necessary for STAMP only and cannot claim to be a complete or

[4]See discussion in **Section 2.2.2**

accurate representation of the theoretical framework of Control Systems. Instead it defines what is sufficient for STAMP and the terms as they are understood in STAMP. This limits the application of the `Control System` module to use with STAMP.

The final module authored is the `STAMP` ontology, which contains those terms necessary for a STAMP analysis that are not already defined. Each of these modules will be discussed in turn from top to bottom.

3.2 Top Ontology Module

This ontology module, depicted in **Figure 3.2**, provides a few common terms typical in top-ontologies (such as BFO (Arp et al. 2015) or UFO (Guizzardi, Botti Benevides, et al. 2021)) in order to orientate the reader and to provide discipline to the authorship of the lower-level ontology modules (Arp et al. 2015).

With regards to the issue of realism, this ontology subscribes to the position of GFO[5]. Therefore it takes as categories conceptual things, such as "System" or "Situation", as equally valid as those that are universals (or natural classes/kinds (Arp et al. 2015, p.14)). It's akin to a relativistic view of truth, rather than absolutist: it represents the world as it is in the mind of the author rather than how it exists external to perception. That said, the terms and structure are relatively close to BFO (Arp et al. 2015), beginning with the two-category division into `Continuant` and `Occurrent`, whereas UFO for example is a four-category ontology (Guizzardi, Botti Benevides, et al. 2021). Continuants are those entities that are conceived to "continue to exist through time" (Arp et al. 2015, p.52), such as this thesisp: it continues to exist through the time when it does exist and it is wholly present for all of that time. Occurrents are those entities that are conceived to "occur, which means that they are spread out not just in space but also in time" (Arp et al. 2015, p.52), such as the process of your reading of this thesis: parts for which have already passed and are spread through the time taken in reading. The sole Occurrent defined is a Process, which is all that is required for this use-case; there is no goal to author a complete top-ontology but only that which is required for the lower level ontology modules.

Continuant includes:

- Entity: a Continuant that doesn't depend on the existence of anything else to exist

[5]See discussion in **Section 2.2.2**

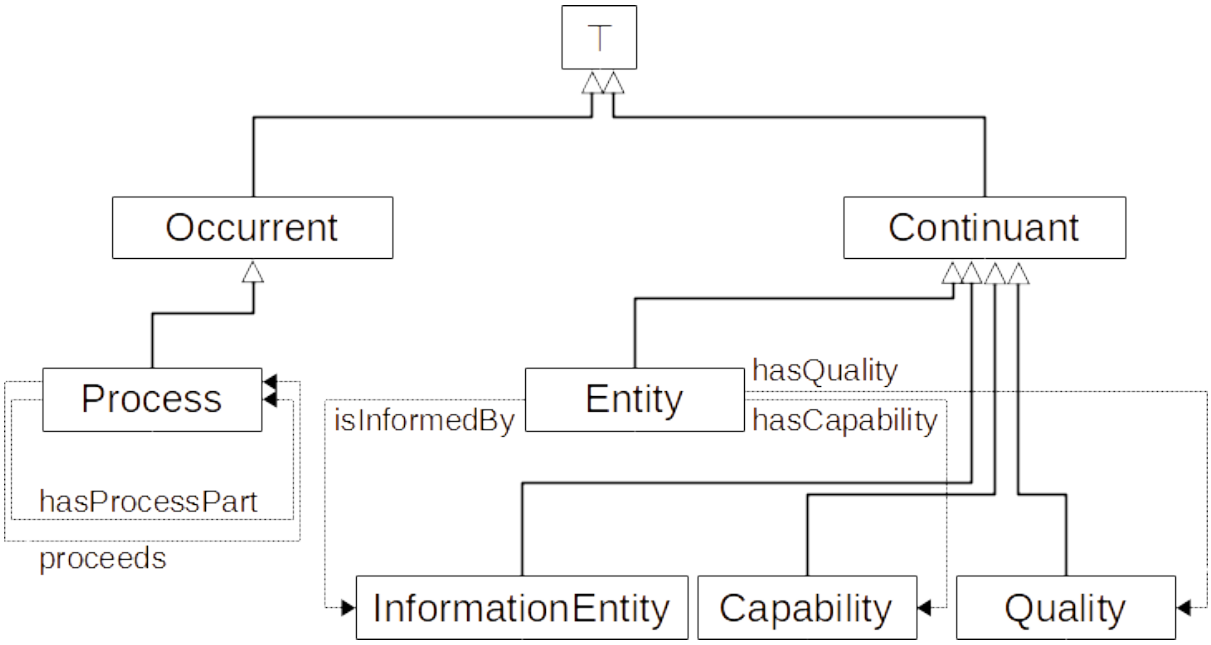

Figure 3.2: Tiny Top Ontology: open arrows depict subsumption, other indicative roles are labeled

- Quality: an attribute or property that inheres in some entity
- Information Entity: the carrier of some information
- Capability: the capability to realise some process that inheres in some entity

3.3 Situation Ontology Module

The purpose of this ontology is to allow reasoning over processes. For STAMP it is necessary to describe the preconditions and consequences of control actions, such as: "Opening the door when the power is on may cause the hazard where the operator is exposed to the power source".

3.3.1 Situation Ontology Background

In "The Theory of Situations" (Barwise and Perry 1983) a situation is described by the infons that it supports. An infon is a relationship between a given number of arguments and a polarity that denotes whether the infon stands in the situation or not. So the situation in which the door is open and a power source is *not* on can be described as:

$$\ll hasQuality, \texttt{door}, \texttt{open position}, 1 \gg,$$

$$\ll hasQuality, \texttt{power}, \texttt{on}, 0 \gg$$

Unfortunately, the ontology derived from "The Theory of Situations" by Kokar et al. (2009) does not represent the transitions between situations. The same is true of DOLCE's 'ExtendedDnS:situation' (Gangemi and Mika 2003), which is defined as a non-agentive social object that satisfies some description. Thus they cannot be used independently to project the consequences of actions to the next situation: a vital requirement for safety analysis with STPA. Additional terms are required to capture sufficient information to reason about future situations, which can be informed by the Situation Calculus, which can be used for this kind of reasoning.

Situation Calculus (McCarthy and Hayes 1969) is used for reasoning about the transitions between situations described by fluents, via actions. McCarthy and Hayes (1969) proposed it as a representation that can answer, albeit not always correctly, questions including: "What will happen if I do a certain action?" (McCarthy and Hayes 1969, p.4) and "What will happen next in a certain aspect of the situation?" (McCarthy and Hayes 1969, p.4). These are questions frequently asked by the safety analyst, and the correctness of the answers matters to them as a wrong answer could result in an incorrect behavioral model of the system, as per the introductory case of the Therac-25 where speed of input actions could result in the machine delivering lethal doses of radiation (Nancy G Leveson and Turner 1993).

Situation Calculus, however, is not consistently defined: authors disagree on what a situation is. Unless otherwise qualified, this thesis defaults to the definition of Reiter (2001) as that one is accompanied with source-code for a reasoner. McCarthy and Hayes (1969) who proposed the representation, defined a situation as a complete description of the state of a universe at any particular time. Given the impossibility of such a complete description, they resigned to using partial descriptions, from which useful information can still be garnered. The descriptors used for those things that can change between situations are called "fluents".

In contrast, Reiter (2001) defines a situation as the sequence of actions taken since some initial situation 's0'. Such a situation carries its history with it, from which the fluents can be determined. Thus the `door open` and `power off` situation under Reiter's definition could be any of

these:

$$s0 \equiv S_{eg}$$
$$do(toggle(power), do(open(door), s0)) \equiv S_{eg}$$
$$do(make(tea), do(open(door), s0)) \equiv S_{eg}$$

Fluents play a similar role to the infon in "The Theory of Situations" (Barwise and Perry 1983), however the manner in which the truth of the fluent/infon holding in a defined situation differs. Rather than use polarity and a relation to the situation, a fluent (Reiter 2001) either includes the situation in which it stands as an parameter, or this is reified via the relation 'holds', which denotes that the fluent stands in that situation:

$$hasQuality(\texttt{door}, \texttt{open position}, S_{eg}) \wedge \neg hasQuality(\texttt{power}, \texttt{on}, S_{eg}) \equiv$$
$$holds(hasQuality(\texttt{door}, \texttt{open position}) \wedge \neg hasQuality(\texttt{power}, \texttt{on}), S_{eg})$$

Situation Calculus also includes the relation "poss", which is true if the given action is possible to do in the related situation. The consequences of actions are determined depending on the definition of Situation Calculus used due to the differing definitions of a situation. Abstractly, actions transition to a situation in which different fluents hold; they are defined in terms of the kind of situations they are possible in and which fluents they cause to hold.

The Frame Problem is present due to the static nature of an ontology i.e. each fluent that holds in a situation will need to be related to that situation. This Frame Problem is mirrored in STAMP, where context tables are generated via the set-theoretic representation for an analyst to review, or software to execute (Thomas 2013), or rules to review (Gurgel et al. 2015). However, in the ontological approach presented here, there is no requirement to enumerate every possible situation due to the formal model of causality provided by this Situation Ontology and subsumption of situations.

As per the implementation by Reiter (2001), the Qualification and Ramification Problems are ignored. Both issues are issues of reasoning: determining when an action is possible and what the resultant state would be, respectively. The information captured in the ontology directly effects the reasoning possible, and so if another solution were known the ontology should capture

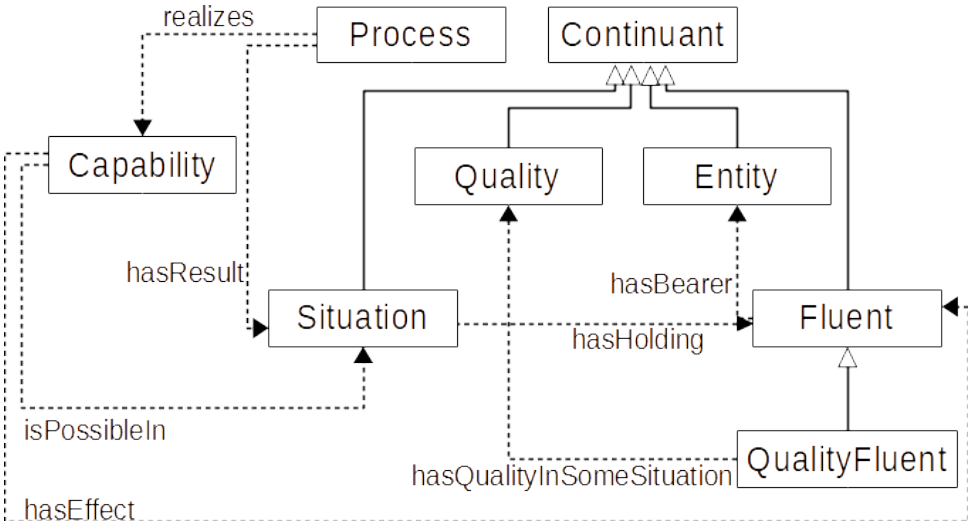

Figure 3.3: Situation Ontology: Derived from the Top ontology, inverse and sub-roles omitted

sufficient information to support it. Furthermore, a discrepancy between the open-world (there may be truth not recorded) and closed-world (all truths are known and recorded) assumptions can arise.

Under the open-world assumption the stated fluents are thus considered as necessary but not complete. Unfortunately under this assumption it's impossible to deduce whether an action is possible in a situation or to project the consequences of actions. If the world is closed, considering the stated fluents as both necessary and complete, then the deduced facts may not be true. However the deductions are comparable to intelligent human thought, given both have the same incomplete information. Therefore a reasoner will not be capable of exceeding human rationality, but it will be able to reach that standard with consistency given the same information and exceed it when given more information than the human has the mental capacity to manage.

3.3.2 Situation Ontology Module Defined

Situation Calculus is an ideal starting point as it supports the reasoning required. However, uncertain terms require deliberation prior to defining. A `Situation` is considered as per "The Theory of Situations" (Barwise and Perry 1983): an abstract situation describing some state-of-affairs in the world. It's defined by the set of fluents that hold in it, as per McCarthy and Hayes (1969). This choice is made as a history of occurrent actions and events can be adequately captured as a `Process`, which in turn can be related to the appropriate situations.

$$\texttt{Situation} \equiv \texttt{Continuant} \sqcap \exists \texttt{hasHolding.Fluent}$$

Situation Calculus typically uses the term "action" to denote the occurrent that transitions from one situation to another, whereas in UFO-B the term "event" is used. In Situation Calculus such actions/events are considered as taking no time, which for the domains it's typically applied to, is untrue. Actions such as "turning on a light" may appear instantaneous, but they're a process of "moving a switch", "completing a circuit" and "passing current through an element" until eventually "light is emitted". Therefore `Process` is used in place of "action" and may be used to subsume actions if a distinction is required to either processes undertaken by an actor, or to subsume events to denote the granularity at which a process will be considered as atomic.

The term `Process` will enable the ontology to record sequences of occurrences and the situations that were transitioned through. However, the STAMP model is required to answer questions with regards to who can do what, and what situation would result if some `Process` occurred. Using these questions the analyst is able to say that in a tailgating situation the front car applying the brakes aggressively could result in a crash and it is the front car driver who can apply the brakes. From this analysis they can then advise on safe driving technique when being tailgated: they know who has control and what can happen that they desire not to occur. This requires constructing hypothetical processes, just like this tailgating example, which can be accomplished for the processes of concern to STAMP and actions of concern to Situation Calculus via additional relations to Capability from the Top ontology module. The realization of a Capability is a Process.

Capabilities in this Situation ontology module are defined with their pre-conditions and post-conditions, via `possibleIn` and `hasEffect` respectively. The range of `possibleIn` is a `Situation` such that all `Situations` it subsumes inherit the relation. The range of `hasEffect` is one or more `Fluents` so that a resulting situation can be deduced from a `Capability` and prior `Situation`, rather than requiring all possibilities to be enumerated. For this `hasEffect` requires two sub-property relations: `hasPositiveEffect` and `hasNegativeEffect` to correlate with Reiter's Simple Solution for determining which fluents hold, as well as unique names axioms for actions (Reiter 2001, p.30-31).

$$\texttt{hasEffect} \sqsubseteq \texttt{topObjectProperty},$$

$$\texttt{hasPositiveEffect} \sqsubseteq \texttt{hasEffect},$$

$$\texttt{hasNegativeEffect} \sqsubseteq \texttt{hasEffect}.$$

A `Fluent` is some relation that holds in a `Situation`. It is reified to a concept to allow it to be `holdsIn` related to `Situation`. No restrictions are placed upon the `Fluent` in this Situation ontology module for portability, however a `hasBearer` relation is provided that may be used to denote the subject of a `Fluent`. The `Fluent` class is also subsumed for use with a `Quality` like so:

$$\texttt{QualityFluent} \equiv \texttt{Fluent} \sqcap \exists \texttt{hasBearer}.\top \sqcap \exists \texttt{hasQualityInSomeSituation.Quality}$$

Figure 3.3 depicts the relations between the three classes. In the Interlock example[6] the door open fluent and the capability to open it can be defined as:

$$\begin{aligned}
\texttt{DoorOpen} &\sqsubseteq \texttt{QualityFluent} \sqcap \exists \texttt{hasBearer.Door} \sqcap \\
&\quad \exists \texttt{hasQualityInSomeSituation.OpenPosition} \\
\texttt{DoorClosed} &\sqsubseteq \texttt{QualityFluent} \sqcap \exists \texttt{hasBearer.Door} \sqcap \\
&\quad \exists \texttt{hasQualityInSomeSituation.ClosedPosition} \\
\texttt{OpeningDoor} &\equiv \texttt{Capability} \sqcap \\
&\exists \texttt{possibleIn.(Situation} \sqcap \exists \texttt{hasHolding.DoorClosed)} \sqcap \\
&\exists \texttt{hasPositiveEffect.DoorOpen} \sqcap \exists \texttt{hasNegativeEffect.DoorClosed}
\end{aligned}$$

This Situation ontology module thus captures sufficient information to be used with a Situation Calculus reasoner, such as the one defined by Reiter (2001). The information can be extracted from the ontology and represented in the syntax of the language required for the reasoner to work with.

3.4 Control System Ontology Module

The purpose of the Control System ontology module, depicted in **Figure 3.4**, is to define control systems as used in STAMP, integrating the Situations module to permit reasoning with Control Actions and Actuations. It's not a complete representation of Control Systems, which is beyond the scope of this work, it's only intended to be sufficient for this STAMP domain.

[6] Included with the STAMP ontology published for public access: Paul S. Brown (Apr. 2022). *PaulBrownMagic/STAMP-Ontology: Initial Release*. Version v1.0.0. DOI: 10.5281/zenodo.6489774. URL: https://doi.org/10.5281/zenodo.6489774

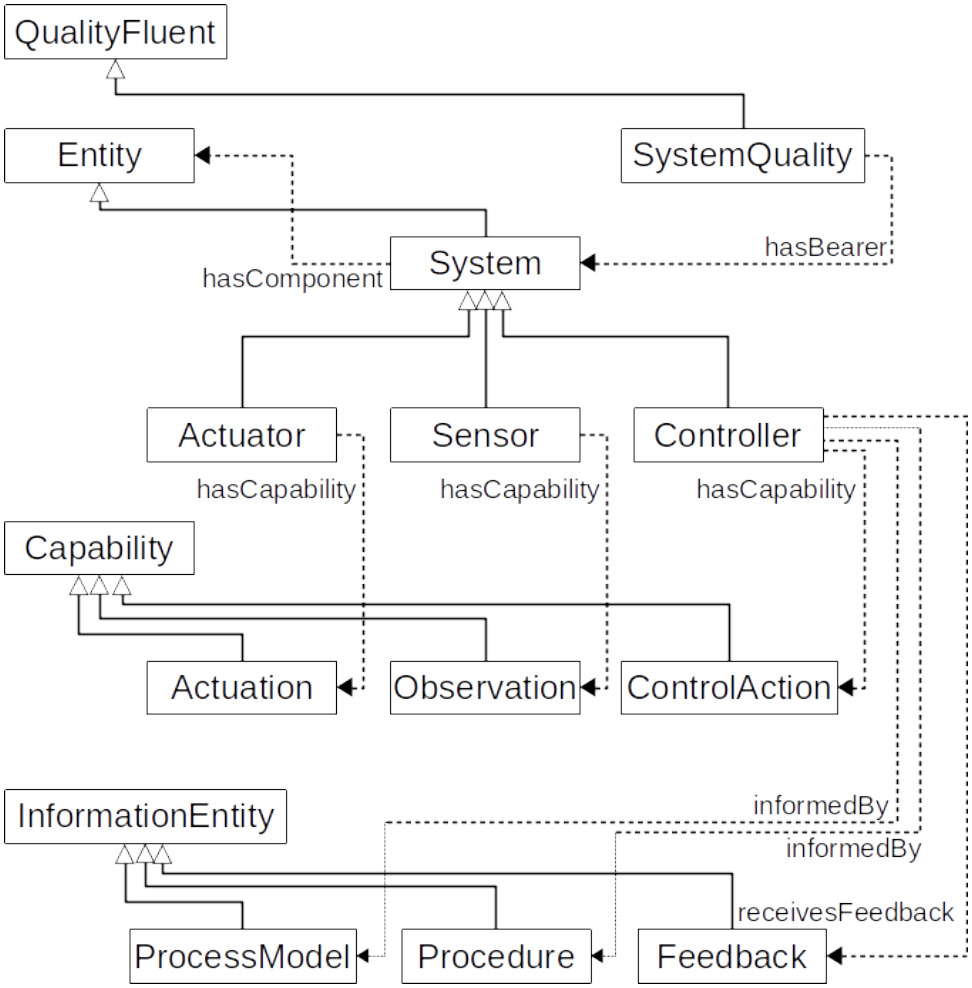

Figure 3.4: Control System Ontology: Subsumption is depicted by open arrows, other labelled roles also form part of their origin concept's definition

This ontology module, describing a certain type of system, requires a definition of `System`, a term used to describe a multitude of diverse things that are represented with the systems model. The word 'system' derives from the Greek $\sigma\acute{\upsilon}\sigma\tau\eta\mu\alpha$ or sústēma, which means: "A whole compounded of several parts or members" (Liddell and Scott 1940). Thus a system is a model of composition, an entity is modelled as a system by defining the relationship between the whole and its constituent part(s). This component-integral relation or functional parthood relation is one in which a part plays a functional role in the whole (Artale et al. 1996; Odell 1994; Vieu 2006; Vieu and Aurnague 2007; Winston et al. 1987). It's this component relation that is used to determine if something is a system:

$$\texttt{System} \equiv \texttt{Entity} \sqcap \exists \texttt{hasComponent}.\texttt{Entity}$$

In a similar manner a sensor or actuator are defined by their purpose: to sense by observation which fluents hold or by actuation cause a fluent to hold respectively. In this definition of a Control Systems ontology module, these kinds of systems are defined by their capability to observe or actuate rather than as a system that realises some observing or actuating. By this definition an instance of some sensor that sits on a shelf and never realises its capability to sense is still a sensor, which eases reasoning about the safety of a system prior to its deployment.

$$\texttt{Sensor} \equiv \texttt{Entity} \sqcap \exists \texttt{hasCapability}.\texttt{Observation}$$

$$\texttt{Actuator} \equiv \texttt{Entity} \sqcap \exists \texttt{hasCapability}.\texttt{Actuation}$$

STAMP makes use of a `Controller` system, which is one that can receive some `Feedback` describing a `Quality` of some entity, and can make use of this information via a Procedure and Process Model, to make decisions regarding its initiation of Control Actions (N. Leveson 2017; Haller et al. 2017). Thus a controller can be defined as:

$$\texttt{Controller} \equiv \texttt{System} \sqcap \exists \texttt{hasCapability.ControlAction} \sqcap \exists \texttt{receivesFeedback.Feedback} \sqcap \exists \texttt{infomedBy.Procedure} \sqcap \exists \texttt{informedBy.ProcessModel}$$

The two different methodologies used with STAMP, STPA and CAST[7], differ in that STPA is undertaken on a conceptual system whereas CAST is undertaken post-incident. In STPA the Control Action considered is the capability to take such an action. For CAST the Control Action considered is the instantiation of some `Process` that realizes this `Capability`, although early in the analysis it may be the `Capability` that is defined prior to determining what Control Actions were actually taken in the Loss Process. Therefore for the purposes of the STAMP domain a `ControlAction` is subsumed by `Capability`, the CAST case being handled by the instantiation of the `Process` that `realizes` the `ControlAction` `Capability`.

These Control Actions, when realised, request some `Actuator` execute some actuation, which is the mechanism by which transition between situations is achieved. There is no guarantee that the actuation will be achieved, nor that an unrequested actuation will be executed, nor that the actuation will be precise. Likewise for the `Sensor` and some observation action. This also differs from reasoning with a Situation Calculus reasoner (extraneous to the ontology) in conjunction with the Situation ontology module, which will treat every action done as successful. This either has to be born in mind by the analyst using such a reasoner (that its aid is incomplete), or additional "control actions" need to be defined to aid the reasoner, such as "close door" and "attempt to close door", which cause different fluents to hold/not hold.

3.5 STAMP Ontology Module

The distinction between the STAMP ontology and the STAMP ontology module is that the STAMP ontology is comprised of the Top, Situation, Control Systems and STAMP modules. The STAMP module is the lowest layer in the dependency graph that defines the terms required

[7] Introduced in **Section 2.1.2**

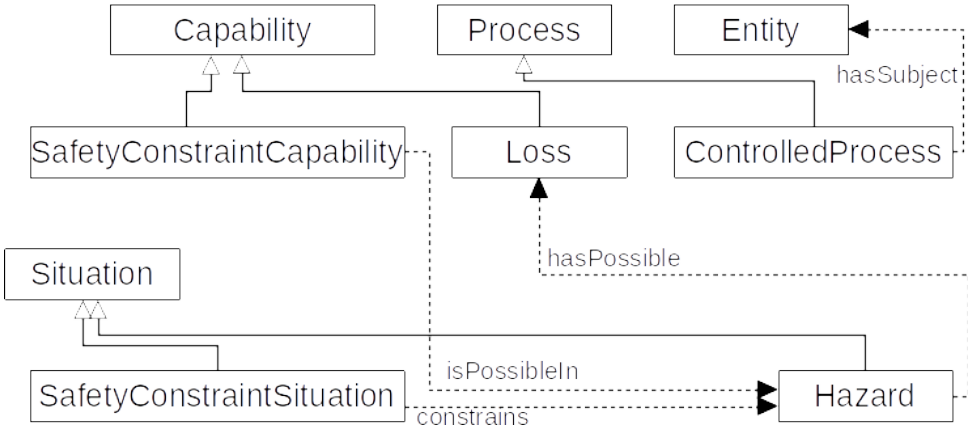

Figure 3.5: Step 1 terms in the STAMP Ontology

to capture the STAMP model that have not already been defined in those modules that it depends upon. While describing the STAMP ontology module the STPA procedure, outlined in **Appendix A**, is followed to contextualise the terms. The fourth step of the 4-step STPA procedure is not described as no new terms are required to capture this phase.

3.5.1 Step 1: Safety Situations

The intention of Step 1 of STPA is to define the scope and purpose of the analysis by identifying the losses, hazards, and safety constraints.

The term "Loss" is used in STPA as an umbrella term for those used in different industries, such as accident or adverse event. It is used to describe that which is to be prevented (N. Leveson and Thomas 2018). As an umbrella term it subsumes concepts such as "Accident" or "Failure", which are not defined here as they are not required for STAMP, which only uses "Loss" in place of these other terms (N. Leveson and Thomas 2018).

This introduces some confusion as in STAMP, as opposed to STPA, an accident is an event that results in a loss, where loss is undefined (N. Leveson 2017) but must be a kind of situation under this ontological defintion. Given the succession of these resources it's taken that STPA "Loss" supersedes the prior one such that a `Loss` is an event which transitions to a situation where something of value to the stakeholders has been lost.

By the Situation ontology, this lost thing is some `Fluent` that holds which the stakeholders do not want to hold. However, the thing that is of value to the stakeholders need not be captured formally in the definitions of `Loss` nor any resulting situation, as it's not required by STPA or CAST, although they can be included as stakeholder goal situations (Rising and Nancy G.

Leveson 2018).

In the definition of "Loss", STPA and CAST diverge. Within a CAST analysis some instance of a loss has actually occured and is being investigated (N. Leveson 2017). In STPA the analyst is concerned with the capability of the system to realize a loss (N. Leveson 2017; N. Leveson and Thomas 2018). As a `Capability` can be related to a `Process` via `realizes`, in this ontology a `Loss` is subsumed by `Capability`. Thus for STPA a `Loss` is defined as used, whereas for CAST it's necessary to define the `Loss` capability and the instance as a realization `Process` of that `Loss`.

A `Hazard` then is a `Situation` in which a `Loss` is possible, described by STAMP as leading to a loss when combined with a set of worst-case environment conditions.

$$\texttt{Loss} \sqsubseteq \texttt{Capability}$$

$$\texttt{Hazard} \equiv \texttt{Situation} \sqcap \exists \texttt{hasPossible.Loss}$$

A "Safety Constraint" has two definitions in STAMP:

- ⟨Safety Constraint⟩ = ⟨System⟩ & ⟨Condition to Enforce⟩ & [⟨Link to Hazards⟩]
- ⟨Safety Constraint⟩ = If ⟨Hazard⟩ occurs, then ⟨what needs to be done to prevent or minimize a loss⟩

These definitions are fundamentally in conflict. The first is a situational definition, describing some fluents that must hold. The second is a process-based definition, describing something to be done in a situation. Therefore, despite being given the same name in STAMP, they are different kinds of things. The first is a subclass of `Situation`, whereas the second is the reification of an if/then relation between a situation and a capability to be realized in order to avoid the related hazard. To avoid confusion that may arise from two definitions for the same term, they are distinguished by suffixing "Situation" and "Capability".

To formally define "Safety Constraint Situation" requires determining what is meant by the two '&'s in the provided definition above. From the Control Systems module, "⟨System⟩ & ⟨Condition to Enforce⟩" are reified into a `QualityFluent`. There may be multiple fluents constituting a safety constraint, therefore it is a kind of `Situation`, which is defined as a collection of fluents that hold/don't hold in it.

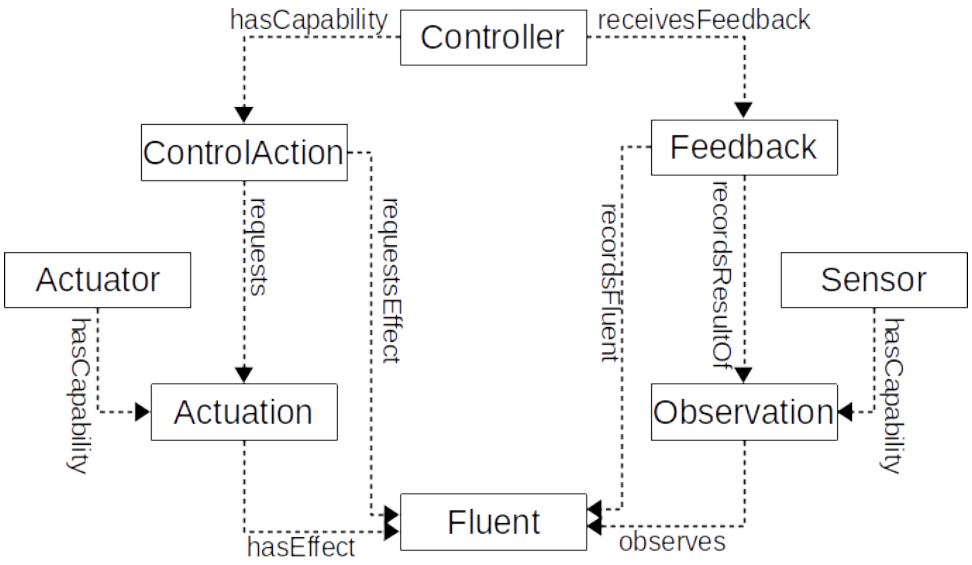

Figure 3.6: System Control Abstraction in STAMP: depicting the control loop defined in the Control System Ontology with the addition of 'requestsEffect' and 'recordsFluent' abstraction roles

The second '&' then is a relation between two `Situations`, in this case the intention is to state that one `Situation` prevents the other, which is to say that the instantiation of one `Situation` at some point in time at some particular location prohibits the instantiation of the other `Situation` at the same time and location.

For example, a situation in which something conceived to be a power source is powered on cannot co-occur with the situation in which there is not anything conceived to be a power source that is powered on. The `constrains` role relating two particular situations is an assertion by the analyst requiring particularly careful consideration as it is asserted with concepts rather than instances.

$$\texttt{SafetyConstraintSituation} \sqsubseteq \texttt{Situation} \sqcap \exists \texttt{constrains.Hazard} \sqcap \neg\exists \texttt{hasPossible.Loss}$$

$$\texttt{SafetyConstraintCapability} \sqsubseteq \texttt{Capability} \sqcap \exists \texttt{isPossibleIn.Hazard} \sqcap \neg\exists \texttt{hasResult.Hazard}$$

3.5.2 Step 2: Control Structure

The purpose of Step 2 is to develop the model of the control structure, including the control actions and feedback, which forms the basis of the remaining analysis. This is depicted in

Figure 3.8 for the interlock example system. This stage of the analysis is conducted at a level of abstraction that conflates the control action with the actuation to not yet require the actuator to be described. The same abstraction/conflation is done for feedback and sensor. This is depicted in **Figure 3.6**, where the 'requestsEffect' role bypasses the actuation. These abstraction roles are defined as compositions:

$$\texttt{requests} \circ \texttt{hasEffect} \sqsubseteq \texttt{requestsEffect}$$

$$\texttt{requests} \circ \texttt{hasPositiveEffect} \sqsubseteq \texttt{requestsPositiveEffect}$$

$$\texttt{requests} \circ \texttt{hasNegativeEffect} \sqsubseteq \texttt{requestsNegativeEffect}$$

$$\texttt{recordsResultOf} \circ \texttt{observes} \sqsubseteq \texttt{recordsFluent}$$

This has the additional benefit of accounting for occasions when the distinction between Controller and Actuator or Sensor is pedantic, such as distinguishing the mind, hand and eye of the Human Controller. With these roles there is no-longer the need to decompose the Human Controller into their component parts, or other Controllers that exert direct control or directly observe. Furthermore, the decomposition isn't prohibited should it be required.

A controlled process, depicted in **Figure 3.5**[8], is one which some controller can influence: in the process there must be some controller that can issue a control action with the intention of changing the fluents related to some system that is a subject of the controlled process. Again this is an abstraction used in STAMP, it ignores the physical structure and instead creates a functional model, which excludes the actuators and sensors during this phase of the analysis (N. Leveson and Thomas 2018). To support this abstraction, a role is required to declare that some systems are the subjects of the controlled process, meaning the change of the qualities they bear in fluents is an intended consequence of the process.

$$\texttt{ControlledProcess} \sqsubseteq \texttt{Process} \sqcap \exists \texttt{hasSubject.System}$$

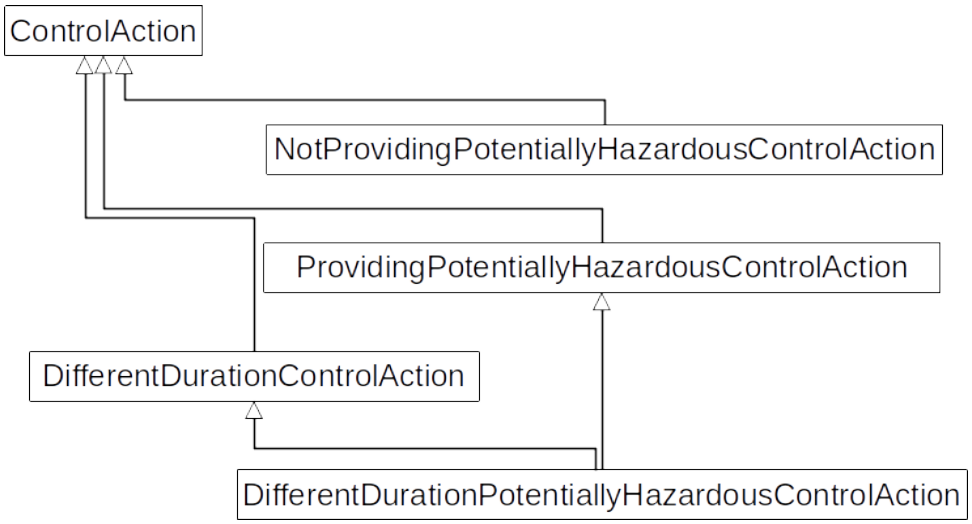

Figure 3.7: A partial view of the taxonomy of Control Actions on STAMP; note the intersection to define DifferentDurationPotentiallyHazardousControlAction

3.5.3 Step 3: Identifying Potentially Unsafe Control Actions

Step 3 is the first of two steps in which safety concerns are identified. In this step those control actions that may result in a hazardous situation are identified and related to the Hazards they may result in.

STAMP considers several manners in which a control action can be unsafe, including providing a control action at a different time or for a different duration. It is claimed that the categories considered are provably exhaustive, although no taxonomy of them is provided (N. Leveson and Thomas 2018), which is remedied in this ontology. In STAMP control actions are defined by their labels, such that braking too early, too late, too long or too short are all the same control action: "brake". However, by the ontological definitions in the Situations (**Section 3.5.1**) and Control Systems (**Section 3.5.1**) modules, a control action is defined by the situation it is possible in and the fluents it causes to hold and not hold in the resulting situation.

A control action done at a different time is possible in a situation different from the defined one. A control action done for a different duration causes different fluents to hold from one done for the expected duration. Therefore these are different control actions by this ontological definition, and need to be defined as such. Fortunately, to aid the analyst, they may all share the same label. Note that the ontological definition does not preclude defining an intersection control action done at both a different time and for a different duration, an intersection not

[8]Controlled Process as depicted in STAMP's Control Hierarchy diagram will also be shown later in the this chapter in **Figure 3.8** when working through an Interlock example

explicitly considered in the STPA Handbook (N. Leveson and Thomas 2018), highlighting a benefit of explicit taxonomy.

$$\texttt{DifferentDurationControlAction} \sqsubseteq \texttt{ControlAction}$$

$$\texttt{TooShortControlAction} \sqsubseteq \texttt{DifferentDurationControlAction}$$

$$\texttt{TooLongControlAction} \sqsubseteq \texttt{DifferentDurationControlAction}$$

$$\texttt{TooShortControlAction} \sqcap \texttt{TooLongControlAction} \sqsubseteq \bot$$

$$\texttt{DifferentTimingControlAction} \sqsubseteq \texttt{ControlAction}$$

$$\texttt{TooSoonControlAction} \sqsubseteq \texttt{DifferentTimingControlAction}$$

$$\texttt{TooLateControlAction} \sqsubseteq \texttt{DifferentTimingControlAction}$$

$$\texttt{TooSoonControlAction} \sqcap \texttt{TooLateControlAction} \sqsubseteq \bot$$

Identifying Control Actions that are potentially unsafe is the key task of this step. Although the ultimate responsibility for identifying these actions lies with the analyst, it is possible via reasoning to provide assistance. To do this the terms "Providing Potentially Hazardous Control Action" and "Not Providing Potentially Hazardous Control Action" need to be carefully defined. A Control Action is potentially hazardous when provided if providing it could transition to a Hazard situation. Whereas not providing it could be potentially hazardous if it could transition out of a Hazard or into a Safety Constraint Situation, or if it's subsumed by a Safety Constraint Capability.

A Situation, which subsumes both Hazard and Safety Constraint Situation, can be defined by the Fluents that hold as well as those that do not. Both the positive and negative roles are accounted for in the definitions of these potentially unsafe control actions:

$$\texttt{ProvidingPotentiallyHazardousControlAction} \equiv$$
$$\texttt{ControlAction} \sqcap$$
$$(\exists \texttt{requestsPositiveEffect}.(\texttt{Fluent} \sqcap \exists \texttt{holdsIn}.\texttt{Hazard})$$
$$\sqcup \exists \texttt{requestsNegativeEffect}.(\texttt{Fluent} \sqcap \neg \exists \texttt{holdsIn}.\texttt{Hazard}))$$

$$\texttt{NotProvidingPotentiallyHazardousControlAction} \equiv$$
$$\texttt{SafetyConstraintCapability} \sqcup (\texttt{ControlAction} \sqcap ($$
$$\exists \texttt{requestsPositiveEffect}.(\texttt{Fluent} \sqcap$$
$$\exists \texttt{holdsIn}.\texttt{SafetyContraintSituation})$$
$$\sqcup \exists \texttt{requestsNegativeEffect}.(\texttt{Fluent} \sqcap$$
$$\neg \exists \texttt{holdsIn}.\texttt{SafetyContraintSituation})$$
$$\sqcup \exists \texttt{requestsNegativeEffect}.(\texttt{Fluent} \sqcap \exists \texttt{holdsIn}.\texttt{Hazard})$$
$$\sqcup \exists \texttt{requestsPositiveEffect}.(\texttt{Fluent} \sqcap \neg \exists \texttt{holdsIn}.\texttt{Hazard})))$$

The potentially unsafe control actions of a different duration and different timing are only considered potentially unsafe when provided; not providing control actions that aren't expected to be provided are also not considered in the analysis. As control actions of different duration or timing are subsumed by Control Action, the potentially unsafe ones are a specialisation of Providing Potentially Hazardous Control Action and can be defined by intersecting the appropriate concepts:

$$\texttt{TooLongPotentiallyHazardousControlAction} \equiv$$
$$\texttt{ProvidingPotentiallyHazardousControlAction} \sqcap$$
$$\texttt{TooLongControlAction}$$
$$\texttt{TooShortPotentiallyHazardousControlAction} \equiv$$
$$\texttt{ProvidingPotentiallyHazardousControlAction} \sqcap$$
$$\texttt{TooShortControlAction}$$

$$\texttt{TooLatePotentiallyHazardousControlAction} \equiv$$
$$\texttt{ProvidingPotentiallyHazardousControlAction} \sqcap$$
$$\texttt{TooLateControlAction}$$
$$\texttt{TooSoonPotentiallyHazardousControlAction} \equiv$$
$$\texttt{ProvidingPotentiallyHazardousControlAction} \sqcap$$
$$\texttt{TooSoonControlAction}$$

The linguistic based definition of Control Actions used in STAMP, where they're identified by a label, can be determined if required by including a label, as exemplified for a braking example:

$$\texttt{BrakingControlAction} \equiv \texttt{ControlAction} \sqcap \exists \texttt{rdfs:label.\{“Brake”\}}$$

3.6 Illustrative Example Use

The interlock system illustrative example used in STAMP educational materials (N. Leveson 2017) is reused here[9] to demonstrate usage and highlight the benefits to the analyst using this ontological approach. This interlock system consists of a high-powered energy source such that being in proximity of it in a powered state poses a risk to one's health. To prevent contact during normal operation the system also has a barrier in the form of a door. During maintenance a human operator is required to open the door in a safe manner. To make this possible, a power

[9]The complete ontology for interlock and STAMP is publically available: Paul S. Brown (Apr. 2022). *PaulBrownMagic/STAMP-Ontology: Initial Release*. Version v1.0.0. DOI: 10.5281/zenodo.6489774. URL: https://doi.org/10.5281/zenodo.6489774

controller is included in the system, which is responsible for managing whether the power source is on or off.

3.6.1 Step 1: Analysis Scope

For the example interlock system the hazard to avoid is having the power on and door open at the same time: in such a situation it's possible for a person to be injured or killed. By enforcing the power to be off whenever the door is open the hazard can be avoided, which in STAMP terms is a safety constraint. The terms for Step 1 STPA without software support would be defined, with links to other terms in square brackets, as:

- L-1: Loss of life or injury to people
- H-1: Power source is powered on and the door is open [L-1]
- SC-1: Power source must be off when the door is open [H-1]
- SC-2: Door must be closed when the power is on [H-1]

L-1 is declared as a subclass of Loss without further ado. The hazardous situation also requires the two fluents that describe the door being open and the power being on hold. Declaring these fluents signposts important components and qualities to be considered in the analysis, avoiding wasting resources on futile definitions and unnecessary complexity. Only those pertaining to the door are included here as the power ones follow the same pattern.

$$
\begin{aligned}
\texttt{L-1} &\sqsubseteq \texttt{Loss} \\
\texttt{H-1} &\equiv \texttt{Hazard} \sqcap \exists \texttt{hasHolding.DoorOpenFluent} \sqcap \exists \texttt{hasHolding.PowerOnFluent} \\
\texttt{DoorOpenFluent} &\equiv \texttt{SystemQuality} \sqcap \exists \texttt{hasBearer.Door} \sqcap \\
&\qquad \exists \texttt{hasQualityInSomeSituation.OpenPosition} \\
\texttt{Door} &\sqsubseteq \texttt{System} \\
\texttt{Position} &\sqsubseteq \texttt{Quality} \\
\texttt{OpenPosition} &\sqsubseteq \texttt{Position}
\end{aligned}
$$

Conducting an analysis is accomplished by extending the STAMP ontology, defining concepts

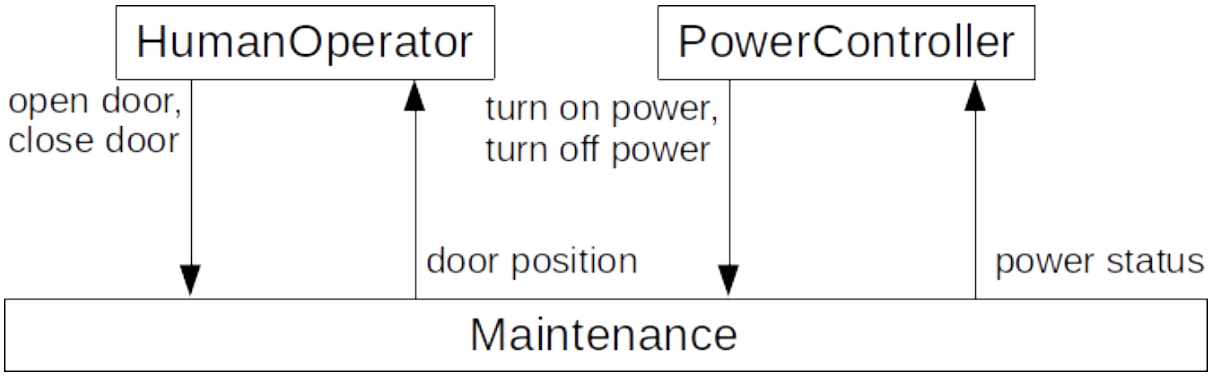

Figure 3.8: Control Hierarchy for Interlock: Human Operator and Power Controller are Controllers controlling the Maintenance Controlled Process with Control Actions denoted by downward-pointing arrows and Feedback denoted by upward-pointing arrows

that are subsumed by those defined within. This enforces a stricter interpretation and more robust definition than STAMP alone. For example, assuming a single door and power source it would be adequate in STAMP to state that the power being off when the door is open prevents the hazard. However, the ontological reading is regarding the concepts rather than instances, so it becomes necessary to state the safety constraint as: "Whenever something that is a kind of door is open, there should be nothing that's a kind of power source that is on". With this second definition a second door or power source can be added, or an additional quality, such as ajar subsumed by open, and the statement will still be true. Instances can be added to the ontology, but given the use of STPA in system design, the required reasoning should not, and does not require them.

Therefore the non-naïve definition of SC-2 is:

$$\texttt{SC-2} \equiv \texttt{SafetyConstraintSituation} \sqcap \exists \texttt{constrains.H-1} \sqcap$$
$$\exists \texttt{hasHolding.PowerOnFluent} \sqcap \neg \exists \texttt{hasHolding.DoorOpenFluent}$$

In Step 1 the ontological definition has made explicit the situations and systems that are pertinent to the analysis by distinguishing them as terms outside a textual description such that they are interpretable by software. The fluents describe key systems and qualities to be controlled, indicating the need for control actions to cause the corresponding fluents to hold, signposting Step 2.

3.6.2 Step 2: Control Hierarchy

The control hierarchy diagram for the interlock system is shown in **Figure 3.8**, it summarises the output of Step 2. Note it's at a level of abstraction where sensors and actuators are excluded and a controlled process, "Maintenance" in **Figure 3.8**, hides other choice details.

The Step 1 fluents indicate how to begin creating this diagram; if the power source can be powered on and not powered on then something must be capable of turning it on and off via some control actions. The control-loop controlled by the Power Controller is defined:

$$\begin{aligned}\texttt{PowerController} \sqsubseteq\ & \texttt{Controller} \sqcap \\ & \exists \texttt{hasCapability.TurnOnPower} \sqcap \\ & \exists \texttt{hasCapability.TurnOffPower} \sqcap \\ & \exists \texttt{receivesFeedback.PowerSourceStatusObservation}\end{aligned}$$

$$\begin{aligned}\texttt{TurnOnPower} \equiv\ & \texttt{ControlAction} \sqcap \\ & \exists \texttt{isPossibleIn.(Situation} \sqcap \exists \texttt{hasHolding.PowerSourceOff)} \sqcap \\ & \exists \texttt{requestsNegativeEffect.PowerSourceOff} \sqcap \\ & \exists \texttt{requestsPositiveEffect.PowerSourceOn}\end{aligned}$$

$$\begin{aligned}\texttt{PowerSourceStatusObservation} \equiv\ & \texttt{Feedback} \sqcap \\ & \exists \texttt{recordsFluent.[PowerSourceOn} \sqcup \texttt{PowerSourceOff]}\end{aligned}$$

In **Figure 3.8** it's the maintenance process that is being controlled by the human operator and power controller. The door and power source are subjects in this procedure; their fluents are subject to effective control actions, and they do not issue control actions themselves. This demonstrates the use of the controlled process abstraction.

$$\texttt{Maintenance} \sqsubseteq \texttt{ControlledProcess} \sqcap \exists \texttt{hasSubject.Door} \sqcap \exists \texttt{hasSubject.PowerSource}$$

3.6.3 Step 3: Generating Potentially Unsafe Control Actions

In this example system, all four control actions can be potentially hazardous, shown here as Unsafe Control Actions (UCA) in the standard STPA format:

- UCA-1: Power Controller provides Turn On Power when Door Position Open [H-1]
- UCA-2: Power Controller does not provide Turn Off Power when Door Position Open [H-1]
- UCA-3: Human Operator provides Open Door when Power Source On [H-1]
- UCA-4: Human Operator does not provide Close Door when Power Source On [H-1]

To declare the `TurnOnPower` as potentially unsafe when not provided, the definition can be changed to:

$$\begin{aligned}\texttt{TurnOnPower} \equiv\ & \texttt{NotProvidingPotentiallyHazardousControlAction} \sqcap \\ & \exists \texttt{isPossibleIn.(Situation} \sqcap \exists \texttt{hasHolding.PowerSourceOff)} \sqcap \\ & \exists \texttt{requestsNegativeEffect.PowerSourceOff} \sqcap \\ & \exists \texttt{requestsPositiveEffect.PowerSourceOn}\end{aligned}$$

In this ontological approach the controller and context situation can be added to create the UCA in STPA format via reasoning. Once defined ontologically, it's possible to recreate this STAMP format through `hasCapability` to the Controller, `isPossibleIn` to the context Situation, and 'providingPotentiallyLeadsTo'/'notProvidingPotentiallyLeadsTo' to the linked Hazards. The `isPossibleIn` Situation requires some reasoning to determine the context in which the Control Action is potentially hazardous as discussed in **Appendix B.1**.

3.7 Reasoning for STPA

One of the advantages of using ontology is that it's machine interpretable and formal (Noy and McGuinness 2001). The reasoning presented here is selected to demonstrate the utility of such reasoning to STPA, as well as the variety of methods that may be employed with this STAMP

ontology. Additional reasoning via sets is included in **Appendix B**.

Subsumption can aid in identification, such as when considering if something is a Loss or a Hazard, an Actuator or Controller, a System or a Controlled Process. As the analyst is extending the STAMP ontology, any definition of a term they define as subsumed from a STAMP ontology term should be justifiable and also true for the subsuming STAMP term.

This can be exploited in Step 2, as on occasion it can be difficult to determine whether a system should be regarded as an actuator or controller, especially in cases where the system has decision making capabilities but may not realistically be in a position to use them or they may not pertain to the actions under consideration.

An example would be a nurse carrying out some instruction from a doctor, as a human they possess decision making faculties, but their capability to exercise them is restricted by the control hierarchy. Thus it must be judged on a case-by-case basis whether to include the nurse as a controller or actuator. If the nurse is a controller they must have a decision making procedure, a process model and feedback to inform their decision making, and the capability to choose to initiate control actions. If the nurse is an actuator in their role, their innate decision making capabilities that arise from being human must be surpressed so that they effectively are expected to only follow orders. To aid an analyst in formulating an argument a classic expert system backward-chaining procedure can be applied to determine if the required roles can be defined for a controller, i.e is the nurse informed by a Procedure and Process Model, able to choose to do a Control Action taking into account Feedback?

Within Step 3, this reasoning can be exploited in the key task of identifying those control actions that are potentially unsafe. To show a control action such as `TurnOnPower` $\sqsubseteq$ `ProvidingPotentiallyHazardousControlAction` it can be shown that TurnOnPower $\equiv$ `ProvidingPotentiallyHazardousControlAction` $\sqcap x$ where x is some as yet unknown term.

Substituting the term definitions this becomes:

$$\texttt{ControlAction} \sqcap$$
$$\exists \texttt{isRealizedIn.(Process} \sqcap \exists \texttt{isPossibleIn.(Situation} \sqcap \exists \texttt{hasHolding.PowerSourceOff))} \sqcap$$
$$\exists \texttt{requestsNegativeEffect.PowerSourceOff} \sqcap \exists \texttt{requestsPositiveEffect.PowerSourceOn}$$
$$\exists \texttt{recordsFluent.PowerStatusFluent} \equiv$$
$$\texttt{ControlAction} \sqcap \texttt{x} \sqcap$$
$$(\exists \texttt{requestsPositiveEffect.(Fluent} \sqcap \exists \texttt{holdsIn.Hazard})$$
$$\sqcup \exists \texttt{requestsNegativeEffect.(Fluent} \sqcap \neg \exists \texttt{holdsIn.Hazard}))$$

There are matching ControlAction terms on both sides, and x can be unified with:

$$\exists \texttt{isPossibleIn.(Situation} \sqcap \exists \texttt{hasHolding.PowerSourceOff))} \sqcap$$
$$\exists \texttt{requestsNegativeEffect.PowerSourceOff} \sqcap \exists \texttt{recordsFluent.PowerStatusFluent}$$

Thus the problem is reduced to showing:

$$\exists \texttt{requestsPositiveEffect.PowerSourceOn} \sqsubseteq$$
$$\exists \texttt{requestsPositiveEffect.(Fluent} \sqcap \exists \texttt{holdsIn.Hazard})$$
$$\sqcup \exists \texttt{requestsNegativeEffect.(Fluent} \sqcap \neg \exists \texttt{holdsIn.Hazard})$$

Only the left of the disjunctive term need be expanded to show the subsumption, and as both sides then share the same role it's sufficient to show that $\texttt{PowerSourceOn} \sqsubseteq \texttt{Fluent} \sqcap \exists \texttt{holdsIn.Hazard}$. `PowerSourceOn` is a `SystemQuality`, which is subsumed by `Fluent`, so the term to match is $\exists \texttt{holdsIn.H-1}$ with $\exists \texttt{holdsIn.Hazard}$. `H-1` is subsumed by `Hazard`, therefore it can be concluded that `TurnOnPower` is subsumed by `ProvidingPotentiallyHazardousControlAction`.

Subsumption is a core task of Description Logics and OWL reasoners, therefore this very useful reasoning can be done efficiently with standard tools.

3.8 STAMP Ontology Discussion

The STAMP Ontology, as defined is capable of capturing the information required for an STPA analysis, with the additional ability to capture further information that supports artificially intelligent reasoning. Although this is demonstrated within the illustrative example use[10], this property of capturing the STAMP model is expected given the notation used in STPA was used and made more explicit when authoring the ontology.

A top-down approach to authoring was followed, which highlighted discrepencies in some terms already in use at the bottom, i.e. STAMP. Notably the term Safety Constraint and the additional intersections of the potentially unsafe control actions. Precision is important in safety analysis, therefore it is hoped this ontological definition will invoke careful consideration of these terms among the experts in the STAMP community.

Following the modular approach has led to two interesting generic modules of this ontology authoring. Due to the requirement to deliberate over hypothetical behaviour, and no such ontology containing sufficient terms to do so, the Situation module was created. This module contains sufficient terms such that sufficient information can be extracted from an ontology to be used with a Situation Calculus reasoner, albeit with the caveats of closing the world and the limitations of Situational Calculus.

The second generic module of interest is the Control System Ontology. Although it's intention is only to describe control systems as understood from STAMP, it provides a reference to those whose research is in the field of defining a comprehensive control systems, or systems, ontology. The STAMP use-case is unusual in that its primary concern is with behaviour, as opposed to the mereology of a system. Consideration of this important aspect of understanding systems will ensure a more comprehensive upper-level ontology being authored for control systems.

With the upper layer, and highly reusable domain layer authored, the remaining authoring is to be undertaken by analysts with their specific cases. This may include analysts who are not experts in STAMP and who are very unlikely to be experts in ontology. Therefore, to support these analyst authors, an ontology authoring environment with support that is specific to STPA has been created. It's expected that underpinning the software with the STAMP ontology will

[10]The whole model is publically available at: Paul S. Brown (Apr. 2022). *PaulBrownMagic/STAMP-Ontology: Initial Release*. Version v1.0.0. DOI: 10.5281/zenodo.6489774. URL: https://doi.org/10.5281/zenodo.6489774

enable intelligent tutoring support enabling the production of a better final product as well as developing the analysts STPA skills.

Chapter 4

Scaffolding Ontology Authoring

In this chapter the Contingent Scaffolding framework (**Section 2.3**) is defined, applied to ontology authoring (**Section 2.4**), and how it's applied to STPA is discussed. The formalisation is done with Situation Calculus (Reiter 2001), which logs the users actions (such as the ontology authoring actions for this use-case). The log of actions then provides a diachronic view of the ontology that can be queried, enabling the Contingent Scaffolding.

4.1 Situation Calculus Notation

The definition of Situation Calculus (introduced in **Section 2.2.2** and **Section 2.4.2**) used is that of Reiter (2001), which is informally summarised here to serve merely as an introduction. Reiter's definitions are used so that his Golog reasoner (Reiter 2001) may also be exploited when creating the application.

Situation Calculus is a second-order language with equality and sorts. The three disjoint sorts are *action*, *situation*, and *object*, where *object* is domain dependent and used for everything that isn't an *action* or *situation*. For this application actions such as assertions and retractions are of the *action* sort. The authors and triples are of the *object* sort. The *situation* sort is a term described shortly.

In addition to actions, situations and objects, the standard logical operators are also used. Within these pages these are:

- $\wedge$ conjunction

- $\vee$ disjunction
- $\neg$ negation
- $\equiv$ equivalence
- $\exists$ existential quantification
- $\forall$ universal quantification

The initial situation is denoted by the symbol $s0$, other situations are built with the $do/2$ term, capturing the sequence of actions taken since $s0$. For example, with *make* and *drink* actions and a *tea* object:

- $s1 = do(make(tea), s0)$
- $s2 = do(drink(tea), do(make(tea), s0))$

When considering what action(s) are possible to do in a situation the predicate $Poss/2$ is used. The interpretation of $Poss(a, s)$ is that *action* a is possible in the *situation* s.

Situations are described by the fluents that hold in the situation, where a fluent is a relation between objects and the situation. For example:

- $hot(tea, s1)$
- $colour(tea, black, s2)$
- $colour(tea, tan, s3)$

For ease of working with these fluents they're often reified with the $Holds/2$ predicate, thus the example fluents can also be written:

- $Holds(hot(tea), s1)$
- $Holds(colour(tea, black), s2)$
- $Holds(colour(tea, tan), s3)$

4.2 Ontology Authoring in Situation Calculus

The ontology authoring framework is defined here with the assumption that the ontology will be defined using triples[1] in implementation. To identify the triple the predicate $spo/3$ is used, named by the initials of: "subject", "predicate,"object". This may be easily adapted according to the requirements of the application.

[1] A data structure containing "subject", "predicate", and "object" as used in the RDF data model (D. Wood, Zaidman, et al. 2014)

Furthermore, the action of asserting or retracting triples in the ontology is considered to be a relation between some reified triple, the one to whom the action is attributed, and some time at which it was executed. This permits detailed analysis of the ontology with regards to who authored what and when. Again, this can be adapted to the meet the requirements of a given application. Fluents to determine the "who" and "when" of assertion and retraction are left to the reader to define as they are minor adaptations of the *asserted*/3 fluent.

First, the axioms for authorship are considered, followed by those for ontological reasoning. The definitions are presented in the format used by Reiter (2001) with the actions and fluents first defined as facts. These are followed by the action precondition axioms defining in what situation it's possible to do an action. Finally the successor state axioms define how the situation is changed by the action from one situation to the next, succeeding situation.

Actions

- $\texttt{assert}(\texttt{spo}(s, p, o), \texttt{author}, \texttt{time})$. Author asserted the triple 's, p, o' at the time.
- $\texttt{retract}(\texttt{spo}(s, p, o), \texttt{author}, \texttt{time})$. Author retracted the triple 's, p, o' at the time.

Fluents

- $\texttt{asserted}(s, p, o, sit)$. The triple '$\texttt{spo}(s, p, o)$' has been asserted in *sit* situation. With reification this is equivalent to $\texttt{Holds}(\texttt{asserted}(s, p, o), sit)$.
- $\texttt{fact}(s, p, o, sit)$. The triple '$s, p, o$' has either been asserted or can be derived through ontological reasoning in *sit* situation. This can also be reified with `Holds/2` as $\texttt{Holds}(\texttt{fact}(s, p, o), sit)$.

Action precondition axioms

$$\texttt{Poss}(\texttt{assert}(\texttt{spo}(s, p, o), author, time), sit) \equiv \neg\exists(a, t)[\texttt{asserted}(\texttt{spo}(s, p, o), a, t) \wedge time > t],$$

$$\texttt{Poss}(\texttt{retract}(\texttt{spo}(s, p, o), author, time), sit) \equiv \exists(a, t)[\texttt{asserted}(\texttt{spo}(s, p, o), a, t) \wedge time > t].$$

Successor state axioms

$$\begin{aligned}
&\texttt{asserted}(\texttt{spo}(s,p,o),\texttt{do}(a,sit)) \equiv \exists(author,time)[\\
&\quad a = \texttt{assert}(\texttt{spo}(s,p,o),author,time)\\
&\quad \vee\texttt{asserted}(\texttt{spo}(s,p,o),sit) \wedge a \neq \texttt{retract}(\texttt{spo}(s,p,o),author,time)],\\
&\texttt{fact}(s,p,o,sit) \equiv\\
&\quad \texttt{asserted}(s,p,o,sit)\\
&\quad \vee\texttt{asserted}(p,hasCharacteristic,reflexive,sit)\wedge\\
&\quad \exists(d)[\texttt{asserted}(p,hasDomain,d,sit) \wedge \texttt{fact}(s,subClassOf,d)] \wedge o = s\\
&\quad \vee\texttt{asserted}(p,hasCharacteristic,symmetric,sit) \wedge \texttt{asserted}(o,p,s,sit)\\
&\quad \vee\texttt{asserted}(p,hasCharacteristic,transitive,sit)\wedge\\
&\quad\quad \exists(z)[\texttt{asserted}(s,p,z,sit) \wedge \texttt{fact}(z,p,o,sit)]\\
&\quad \vee\exists(i)[\texttt{asserted}(p,hasInverse,i) \wedge \texttt{fact}(o,i,s,sit)]\\
&\quad \vee\exists(b)[\texttt{asserted}(b,subPropertyOf,p) \wedge \texttt{fact}(s,b,o,sit)]\\
&\quad \vee\exists(c)[\texttt{asserted}(s,instanceOf,c) \wedge \texttt{fact}(c,p,o,sit)]\\
&\quad \vee\exists(c)[\texttt{asserted}(s,subClassOf,c) \wedge \texttt{fact}(c,p,o,sit)]
\end{aligned}$$

A number of typical rules used to derive additional knowledge are demonstrated above, including inheritance, reflexive, symmetric, transitive and inverse properties. The check on the domain for a reflexive property ensures that the subject is in the set of things that are reflexively related to themselves. These rules may be adapted to the chosen language and features required.

The initial situation, $s0$, can have fluents already holding in it. This is used in conjunction with the STAMP ontology in the software application to reduce the number of actions in the situation that need to be queried over. Therefore in the $s0$ provided to the analyst using the software the STAMP ontology is already defined and ready for them to extend with their use case. Fluents can be defined as holding in $s0$ like so:

$$\texttt{asserted}(\texttt{spo}(s, p, o), s0) \equiv$$
$$s = Hazard \wedge p = type \wedge o = Class$$
$$\vee s = Hazard \wedge p = subClassOf \wedge o = Situation$$

Additional actions can be defined to capture the authors interaction with the ontology authoring tool in order to facilitate appropriate support, an example is given below.

Action

- *focus(area, author, time).* The area, which is some space in the graphical user interface that may or may not be visible to the user at any particular time, entered the authors focus at this time.

Fluent

- *focused(area).* The area is in the authors focus.

Action precondition axiom

$$\texttt{Poss}(\texttt{focus}(area, author, time), sit) \equiv \neg\texttt{focused}(area, sit).$$

Successor state axiom

$$\texttt{focused}(area, \texttt{do}(a, sit)) \equiv \exists(author, time)[$$
$$a = \texttt{focus}(area, author, time)$$
$$\vee\texttt{focused}(area, sit) \wedge \neg\exists(area')(a = \texttt{focus}(area', author, time))],$$

Appendix E contains logs from the user study that show the interspersion of ontology authoring actions with navigation and other actions. How this information is used is the subject of the upcoming **Section 4.3.2**.

4.2.1 Answering Situational Questions

With the defined ontology authoring framework the "Who", "What", "When" and "Where" questions become fluents and follow the same query pattern as the *asserted*/3 fluent. However, with this framework it is also possible to query retractions, some past situation, and all past situations. This is demonstrated by answering a selection of example questions.

- Has this fact been asserted and retracted multiple times? Perhaps a sign of confusion or conflict.

$$\begin{aligned}\exists(s, s', s")(s \sqsubseteq do(\mathtt{T}, s0) \wedge s' \sqsubset s \wedge s" \sqsubset s' \wedge \exists(a, a', a", t, t', t")[\\ asserted(spo(CA-1, subClassOf, ControlAction), a, t, s) \wedge \\ \neg asserted(spo(CA-1, subClassOf, ControlAction), a', t', s') \wedge \\ asserted(spo(CA-1, subClassOf, ControlAction), a", t", s")]]\end{aligned}$$

  This example queries for a retraction and queries past situations.

- Has this triple always been asserted?

$$\begin{aligned}Holds(asserted(Hazard, subClassOf, Situation), s0) \wedge \\ \neg\exists(A) \in \mathtt{T}(\exists(a, t)[A \neq retract(spo(Hazard, subClassOf, Situation), a, t)])\end{aligned}$$

  This example queries over the log, which depending on the implementation could be reduced to two look-ups.

- Has this fact always been true?

$$\forall(s)(s \sqsubseteq do(A, s0) \supset fact(Hazard, subClassOf, Situation, sit))$$

  This example is included as `fact` is not a context-free fluent and thus can't take advantage of the same efficiency benefits the asserted fluent can.

4.3 A Contingent Scaffolding Framework

Contingent Scaffolding, abbreviated to Scaffolding, is a pedagogical strategy defined by D. Wood, Bruner, et al. (1976). This is the method of providing support discussed in **Section 2.3**.

This section will define a formal framework for Contingent Scaffolding where the interactive interventions subscribe to the philosophy of nudges to mitigate the ill-defined nature of the domain.

4.3.1 Interactive Nudges

When contingent scaffolding, the intelligent tutor intervenes into a process via prompts, questions, and physical actions. Due to the nature of scaffolding in an ill-defined domain and operating under the open-world assumption, it must be acknowledged that an analyst can author a valid ontology extension without adhering to the guidance of the tutor. Therefore the tutor proffers its advice, which may be rejected, negotiated, or accepted by the analyst. The negotiation is to be understood as the method by which the analyst can ask for the level of intervention to be increased, albeit the tutor may not always oblige.

Such an perspective on the nature of the interventions is a libertarian-paternalistic one. Libertarian because it's accepted that the analyst is free to respond to the intervention however they wish, including ignoring it. Paternalistic because the intervention is offered with the belief, correct or not, that to follow it would be best for the analyst.

Libertarian-paternalistic actions taken to alter behaviour, such as how interventions are used within this framework, are coined "nudges" by Thaler and Sunstein (2009). The "interactive" descriptor is added to make explicit the negotiation, which is inherent in the scaffolding process of leveling the intervention, as well as the active, thoughtful contemplation of the choice to be made by the analyst.

Nudges may be employed to modify behaviour by appealing to the unconscious aspects of choice-making. This aspect is relevant to the process of ontology authoring; the presentation of the UI can be used to discourage attempts at fruitless work such as trying to complete STPA step 3 before step 2. These kind of nudges are passive rather than interactive. Furthermore, this is the domain of UI-design rather than interventions, such as prompts and questions that are taken in order to provoke thought. It's impossible to present a UI without the design of it having some effect on a user, therefore the design is shared, but it's not the focus of this study.

Contingent Scaffolding uses the term "intervention", and therefore this work will continue to use the same term. However, how interventions are used and thought of is informed by nudges in the defined framework for Contingent Scaffolding and its later application.

4.3.2 Scaffolding Framework Defined

To qualify as contingent scaffolding, each of the three principals H. Wood and D. Wood (1999) need to be covered:

- Help is provided expeditiously when the learner is in trouble
- Help is increased as the learner requires until the solution is reached
- As the learner succeeds, support is withdrawn: a process called "fading"

Identifying when an analyst is in trouble will be accomplished through defining troublesome situation terms. The analyst's situation can then be checked if it's subsumed by some troublesome situation. To accomplish this first requires defining what is meant by "trouble".

The second two points relate to levels of support. Firstly the levels need to be enumerated, secondly the appropriate level needs to be selected. In order to select the appropriate level some record of previous interventions is required. The use of Situation Calculus here not only provides this record, but previous situations can also be queried to determine if an analyst has previously succeeded by resolving an intervention, and thus support may be withdrawn.

To define "trouble", the methodology of STPA that this research is seeking to support is applied: "trouble" is the hazardous situation to be avoided. Considering the taxonomy of Potentially Unsafe Control Actions defined, there are two kinds of unsafe control actions the analyst could take to arrive in trouble: Providing or Not Providing. If an analyst is Providing some action that results in a "trouble" situation, then it is supposed that taking this action was a mistake. As per the libertarian framing it may not actually be a mistake, but within the paternalistic one it shall be called as such. A mistake in this case ranges from critical, such as asserting logical inconsistencies, to advisory.

The analyst Not Providing some action can only be identified by tracking their progress through some process and noting what should have been done according to some intervention definitions. Again, as per the libertarian framing it may not actually be missing, but within the paternalistic one it shall be called as such. These missing actions can range from critical, such as not defining any control actions before trying to determine which ones are potentially unsafe, through to advisory, such as adding thoughtful descriptions or comments.

Thus interventions are separated into two categories: "mistake" and "missing". These two pred-

icates for interventions follow these templates:

$$mistake(\langle \texttt{kind} \rangle, \langle \texttt{situation query} \rangle, \langle \texttt{leading question format} \rangle)$$

$$missing(\langle \texttt{step} \rangle, \langle \texttt{kind} \rangle, \langle \texttt{situation query} \rangle, \langle \texttt{leading question format} \rangle)$$

The actual interventions used in the software application are included in **Appendix C**. The meaning of each parameter will be discussed in turn.

A mistake can be made at any time, whereas missingness can only be determined in relation to progress. Therefore within the term to describe a "missing" intervention a parameter for when to begin checking it is required, denoted by $\langle \texttt{step} \rangle$. As the user progresses through their analysis they will navigate to steps 1-4, so at step 3 the *missing* interventions for steps 1 and 2 are checked.

The parameter is denoted as $\langle \texttt{situation query} \rangle$ is a situation used as a query to be run over the analyst's log in order to determine if the intervention is appropriate. This query is the "trouble" situation, which if subsumed by the analyst's current situation (i.e. is true) will be used to cause the intervention to be executed. In practice this is done with a Situation Calculus query due to the representation of situation in this framework as a history of actions rather than as the collection of fluents that hold in it, as a situation is represented in the STAMP ontology. An example query from **Appendix C**:

```
logged_assertion(Hazard, subClassOf, 'Hazard') and
not (logged_assertion(Loss, subClassOf, 'Loss') and
logged_assertion(Hazard, hasPossible, Loss))
```

There may be more than one instantiation of the situation query if, for example, an analyst repeatedly fails to assert what losses are possible in their hazard situations, as per the example query above. In such a case the parameter in the query that identifies the hazard without a possible loss would be different, but the kind of "missing" is the same and so the intervention level scaffolded should be related to the previous one. For example, the above query could succeed at different times with the bindings:

```
logged_assertion('H-1', subClassOf, 'Hazard') and
not (logged_assertion(Loss, subClassOf, 'Loss') and
```

```
logged_assertion('H-1', hasPossible, Loss))
```

...and later:

```
logged_assertion('H-2', subClassOf, 'Hazard') and
not (logged_assertion(Loss, subClassOf, 'Loss') and
logged_assertion('H-2', hasPossible, Loss))
```

The same query has succeeded in two different ways. The difference is expressed in the query itself. The similarity is the different of kinds of "trouble", which is identified through the ⟨`kind`⟩ parameter, which then also enables multiple predicates to be defined with different queries or at different steps to still be related as the same "kind". This parameter also serves as the lowest-level scaffold offered. The corresponding ⟨`kind`⟩ parameter to the example query is: 'Hazards will lead to a loss in some worst-case environment'.

The final parameter is the ⟨`leading question format`⟩. In the scaffolding levels one scaffold is to ask a leading question, which in theory can be created from the query, but in practice resulted in poor questions. Therefore this additional parameter was added, which is a template for the leading question that can be instantiated with the results from the query. The running example leading question format would substitute the '∼s' for the `Hazard` that the query succeeded with (such as H-1 or H-2):

```
'Have you not asserted which Loss is possible to occur in the
∼s Hazard?'-[Hazard]
```

Next the levels of intervention need to be defined. Although there's no canonical set of levels defined, the six levels of Daniels (2010) can be generalised. This generalisation is required as the software has no verbal capabilities. Additionally, determining the analogue to "materials" from the Daniels (2010, p.109) levels "3: indicates materials" and "4: prepares materials" is also ambiguous within a graphical user interface where there are no physical, material items. Therefore the definition of "physical" is stretched to include the components of the graphical interface, which can be presented to a user or changed by the software to add or remove content.

The levels of intervention should increase in intrusiveness and specificity as the levels also increase. Therefore the interventions transition from prompt, to instruction, to physical action. At each level of action a kind of action to take is given based upon the comparative study of

successful human tutors (Merrill et al. 1992).

- Level 0: no assistance
- Level 1: prompt (interjection prompting reflection)
- Level 2: strategic instruction (leading question)
- Level 3: tactical instruction (step-by-step guidance through target actions)
- Level 4: physical preparation (completes preparatory actions but not the target action)
- Level 5: physical completion (completes target action)

When tutoring in an ill-defined domain it's frequently not possible to offer a level 5 intervention as the correct answer isn't known. To circumvent this difficulty the level 5 intervention is conflated into the level 4 intervention, such that an analysis of the situation is used to determine if a level 5 intervention is possible, or if level 4 must be used. If a level 5 intervention is possible that action is taken in place of level 4 so that level 4 can be treated as the maximum level for the purposes of calculating the level of intervention to offer.

Level 0 can be made implicit due to the use of a query determining if an interactive nudge should be given. If the query fails, then no assistance is to be offered, which is the equivalent of level 0. If the query succeeds then it must be determined what level among the rest the interactive nudge should be pitched at.

Using the defined "missing" and "mistake" interventions, at level 1 the ⟨`kind`⟩ parameter is used as a prompt. At level 2 the ⟨`leading question format`⟩ is populated with the case-specific information determined by the query. At level 3 the ⟨`situation query`⟩ is examined. The first failing clause that the user is both permitted to change and capable of changing is parsed into an instruction[2]. At level 4 that first failing clause's subject part of its `spo/3` triple is used to determine the part of the GUI to bring the user to, which is the physical preparation. If the predicate and object of the triple in the clause are also both known then the software can assert or retract them on behalf of the analyst.

If an intervention is possible in a situation, then that intervention could be done:

$$\exists(i \in Intervention, s \in Situation)[poss(i, s) \subset do(i, s)]$$

[2]In application this is done by via a rule-set generating an instruction that the GUI can execute. What to instruct is determined by what parameters are ground in the first failing clause

It is defined as an implication such that the intervention need not be done. For example, if it were decided to only present one intervention at a time and another were already underway, then even though more may be possible, they may also be ignored by some other rule.

Once an intervention has been elected for `do/2`, it's necessary to determine the level of intervention to offer, assemble any required payload of messages and perhaps physical intervention instruction, before presenting it to the user in the chosen manner for their consumption[3].

The level of intervention is determined by examining the situation term. The level 4 and level 5 interventions are conflated into level 4, due to the inability to always provide a physical intervention[4] where the ontology authoring actions are taken by the software. and preclusion of level 0 by the situation query failing leaves the following levels to choose from:

- Level 1: a general verbal prompt pertaining to the kind of issue;
- Level 2: a leading question pertaining to a specific issue;
- Level 3: an specific instruction pertaining to a specific issue;
- Level 4: physical intervention following the prior instruction as far as possible to do correctly;

As soon as it is possible to first `do/2` an intervention it's considered to be at level 1. In order to increment the level of intervention the user is provided with a "request help'' action, which they can use to increase the level of intervention until they reach the limit at level 4. To decrement the level the situation is searched to find a moment where the intervention was possible, but the next action made it no longer possible. At this moment the user has resolved the issue, and so support is faded. The mechanism for fading is captured in the definition for `intervention/3` below.

The final action provided to the user to affect the intervention level is to "dismiss intervention'', which resets the level to 1. This action is provided in deference to the users' expertise as the AI works with imperfect knowledge. This gives the following definitions to change and determine intervention level:

[3] In application this is done with a rule-set that given the intervention term and a level formats the information to be sent at that level into JSON. The exact details depend heavily on the stack of technologies and languages in use.

[4] Physical as in the software makes changes to the state of the software rather than request the analyst to do so

$$\texttt{Poss}(\texttt{requestHelp}(i), sit) \equiv \texttt{Poss}(i, sit)$$

$$\texttt{Poss}(\texttt{dismissIntervention}(i), sit) \equiv \texttt{Poss}(i, sit)$$

$$\begin{aligned}
&\texttt{interventionLevel}(l, i, s) \equiv \\
&\quad \exists(i \in Intervention, s \in Situation, l \in \{0,1,2,3,4\})[\\
&\quad\quad (s = s0 \wedge l = 1) \vee s = do(a, s') \wedge \texttt{interventionLevel}(l', i, s') \wedge [\\
&\quad\quad\quad (a = \texttt{requestHelp} \wedge l = l' + 1) \vee \\
&\quad\quad\quad (a = \texttt{dismissIntervention} \wedge l = 1) \vee \\
&\quad\quad\quad (\neg\texttt{Poss}(i, s) \wedge \texttt{Poss}(i, s') \wedge l = l' - 1) \vee \\
&\quad\quad\quad (a \neq \texttt{requestHelp} \wedge a \neq \texttt{dismissIntervention} \wedge \\
&\quad\quad\quad\quad \neg(\neg\texttt{Poss}(i, s) \wedge \texttt{Poss}(i, s')) \wedge l = l']]
\end{aligned}$$

4.4 Application to STPA

The Contingent Scaffolding definitions and Ontology Authoring ones are independent, however they can be used in conjunction with one another. With the STAMP ontology defined and the Ontology Authoring defined, no additional definitions are required to enable an STPA analyst to use these in extending the STAMP ontology to their own model.

To apply the Contingent Scaffolding definitions to STPA requires defining situations where the the user has potentially missed something or potentially made a mistake. In situations that subsume these "trouble" situations the intervention will be offered at the deduced level.

These situations were gleaned from the STPA Handbook (N. Leveson and Thomas 2018), extracting the advice given and writing situation queries that could identify when they're contradicted. The interventions identified were organised into `missing` and `mistake` categories:

1. *Mistake:*
    a. Hazards should not include ambiguous or recursive words like "unsafe", "unintended", "accidental", etc. (A situation where one of the aforementioned words is in a hazard label)

b. Avoid using ambiguous and vague labels in the control structure: "commands", "feedback", "status", "computer" (A situation where one of the aforementioned words is in a control action or feedback label)

c. If you have more than 7 hazards, consider grouping or combining them to create a more manageable set (A situation where Hazard has more than 7 subclasses)

d. Fluent can't both hold and not hold in a situation

e. Control Action can't both cause a fluent to hold and not hold

2. *Missing:*

i. *Step 1:*

a. Step 1 precedes Step 2 (A situation in step 2 where no loss or hazard is defined)

b. Hazards will lead to a loss in some worst case environment (A situation where there's some hazard that is missing the "has possible" some loss relation)

c. Hazards must describe states or conditions to be prevented (A situation where a hazard has been defined with no related fluents to describe it)

ii. *Step 2:*

a. Step 2 precedes Step 3 (A situation where definitions for controllers having control action capabilities and feedback are lacking)

b. Check that every controlled physical process is controlled by one or more controllers

c. Check that control actions needed to satisfy the responsibilities are included (A situation where some fluent in a hazard isn't effected by some control action)

d. Check that feedback needed to satisfy the responsibilities is included (A situation where some fluent in the hazard isn't sensed by some controller)

iii. *Step 3:*

a. Ensure traceability is documented to link every unsafe control action with one or more hazards (A situation where a an unsafe control action isn't associated with any particular hazard)

b. Ensure every unsafe control action specifies the context that makes the control action unsafe (A situation where an unsafe control action doesn't have the "is possible in" situation defined)

c. Ensure the unsafe control action contexts are defined clearly (A situation where the "is possible in" situation doesn't have any fluents defined to describe it)

d. Identify all Providing Potentially Hazardous Control Actions

e. Identify all Not Providing Potentially Hazardous Control Actions

Interventions for Step 4 of STPA are not included as Step 4 is not included in the final user evaluation. This decision was made in deference to users' time commitment with regards to both learning STPA and completing an analysis. An additional step solely for the purpose of checking step 3 is used though. This additional step allows the software to look for the things missing from step 3 when the user moves to it, declaring that they believe they're done with step 3.

The definitions of these interventions are best understood by referring to the Prolog term definitions, which are included in **Appendix C**.

4.5 Scaffolding Ontology Authoring Conclusions

In this chapter an ontology authoring framework has been defined in Situation Calculus. This framework provides a diachronic view of the ontology and can be further enriched with non-authoring actions that an application developer deems important. The information captured by this ontology authoring framework provides information that may be used to support an author, including by employing Contingent Scaffolding.

A Contingent Scaffolding framework was also defined using Situation Calculus. This framework is independent of the ontology authoring framework and may be used independently of it. However, when doing so the level 4 and level 5 interventions will require careful consideration to determine what physical interference can be done in some other domain.

Finally, the ontology authoring framework and Contingent Scaffolding framework have been applied together to the STPA domain. This is achieved by defining a set of interventions that can be used to support an analyst as they extend the STAMP ontology.

There is a practical concern with using Situation Calculus when implemented in software. Fluents that aren't context free (See **Section 4.2.1**) need to traverse the log multiple times: once for each "context" fluent that they depend on. When the logs get longer, this could cause the application to run sufficiently slowly that the interventions are not perceived as being immediate. To test if this is the case, these libraries are implemented as software and a user-study conducted from which timings are measured (See **Section 5.7**) in the next two sections.

Chapter 5

Software Implementation

With theoretical definitions of the STAMP ontology, Situation Calculus Ontology Authoring framework, and Situation Calculus Contingent Scaffolding framework, an implementation in software is required to determine the practicality and applicability of these frameworks. How efficacious the theory and implementation are will be assessed in the next section. The key aims are:

- sufficient information captured for intervention
- immediate intervention
- generalized, reusable framework libraries (for other domains)

Prolog was chosen as an implementation language from the outset due to using both Ontology and Situation Calculus as foundations in the design. The application contains approximately 10,000 lines of Logtalk source code plus the GUI written in ClojureScript. The parts authored in Logtalk cover persistence of user projects, situation calculus reasoning, ontology authoring, contingent scaffolding, web server, and generating HTML including forms based upon the history of a user project.

5.1 Software Architecture

The software architecture was intentionally designed to follow the same layered framework as the ontology, as depicted in **Figure 5.1**. To aid in organising and navigating the complex software a naming convention centering around Whitby Abbey(Bede et al. 2008) was adopted; Oswin happens to be the name of a King who was supplanted by the founder of the Abbey,

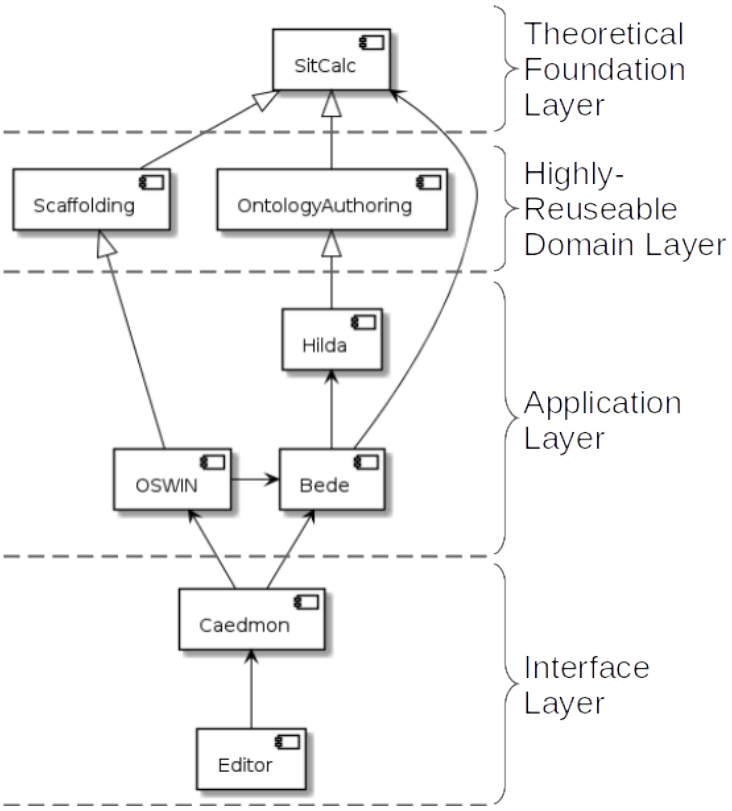

Figure 5.1: Whitby Architecture: open arrows denote extension, closed arrows denote dependence

King Oswiu, whose kingdom also included the land where Leeds University stands today. The responsibilities of the named components are:

- **Hilda**: The wise. Responsible for reasoning over the logs and answering situational queries. This meets the querying requirements for both the ontology authoring and Contingent Scaffolding frameworks.
- **Bede**: The historian. Responsible for the logging of all actions on a per user basis and retrieval of those logs as required. This meets the persistence requirements for the ontology authoring framework.
- **OSWIN**: **O**ntology-driven **S**caffolding **W**ith **I**nteractive **N**udges. Responsible for the contingent scaffolding interventions, which are informed by the paternalistic-libertarian perspective of nudges. Accomplishes the Contingent Scaffolding framework with the exception of determining if a situational query is true for some situation, which is done by Hilda.
- **Caedmon**: The poet. Responsible for the interaction between user interface and the components in the Application Layer.

- **Editor**: The Graphical User Interface.

The personification of the components is intended to ease in reasoning about how the application is working. Thus, following this architecture in **Figure 5.1**, when a user takes some significant action in the Editor, it informs Caedmon of the action taken. Caedmon sends that information to Bede, who will check with Hilda that the user is permitted to take that action based upon the log Bede holds. If it's possible for the user to do the action Bede will update the log and tell Caedmon to acknowledge the action to the Editor.

Caedmon, upon receiving the instruction to acknowledge the action is aware that the situation has changed, and so checks with OSWIN for any interventions it would like sending to the Editor too. OSWIN will ask Bede to give Hilda the user's log and check if one of its intervention queries succeeds for that log. If Hilda finds some intervention query that does succeed, then OSWIN will formulate the intervention payload to be sent to the editor dependent upon how that query succeeded and the current level of that intervention. Bede will also record the intervention in the log at the behest of OSWIN.

The components in the upper 2 layers are used by the Application Layer as third-party libraries, which are generalized. The `SitCalc` library is a dependency of the ontology authoring framework discussed in **Section 4.2** and the Contingent Scaffolding framework discussed in **Section 4.3**. The `Scaffolding` library corresponds to the Contingent Scaffolding framework, and `OntAuth` corresponds to the ontology authoring framework. All three of these libraries can all be, and have been, applied to other domains, which is a key aim to demonstrate their generalisability.

5.2 Refactoring for Reusable, Generalized Code

The entire application has been re-written twice to overcome the architectural issues whose solutions were unsatisfactory. The ramifications of these solution-compromises grew with the complexity of the application. In the final rewrite the code base was transitioned to Logtalk and sufficiently decoupled as to extract three libraries. This extraction was the main motivation behind the rewrites: the prior version was functioning but useful parts of it couldn't be shared.

In this section, the software engineering principles that guided the application rewriting to extract the three libraries are presented, the architectures of the logic programming parts of the last two versions of Whitby are compared, the motivations behind the transition from Prolog to

Logtalk are discussed, and the lessons learned are summarized.

5.2.1 The Dependency Inversion Principle

In refactoring the architecture, the *SOLID principles* of clean architecture (Martin 2018) are applied. These principles are the summary of 20 years of debate between developers attempting to abstract what made their software maintainable and extensible. They are intended to avoid the situation, observed in even market-leading software, where progress is slowing while cost per line of code increases, all while increasing development staff (Martin 2018).

Appropriate application of these principles produces code that is easy to read, maintain, extend and test. These benefits are primarily to those developing and maintaining the software, which then has implications to organisations producing and consuming the software over a period of time. Software with a clean architecture is argued to be easily extensible with new features, typically using a plugin architecture, and robust to changes in business rules, technology, and deployment scenarios (Martin 2018). Thus it reduces application risks and development costs.

The principles of SOLID architecture are:

- **S**ingle Responsibility Principle: each part has one and only one reason to change; it is accountable to one stakeholder
- **O**pen-Closed Principle: code should be open to extension and closed for modification; such as in a plugin architecture where new features are created by adding new code rather than editing existing code
- **L**iskov Substitution Principle: parts should be interchangeable, which makes it robust to even significant changes such as to business rules meeting new legal requirements, or to swapping components such as the database used or GUI framework
- **I**nterface Segregation Principle: do not depend on things not used, which makes dependents of some part robust against changes required by other dependents of that part
- **D**ependency Inversion Principle: high-level policy should not depend on low-level details, but details should depend on policies, such that code which is volatile is not depended upon by code that is stable

The Dependency Inversion Principle, depicted in **Fig 5.2**, is the main principle driving this refactor and the technique used for decoupling. Closely related to this principle is the *code for interface, not for implementation* best practice: no concrete module should be imported into any

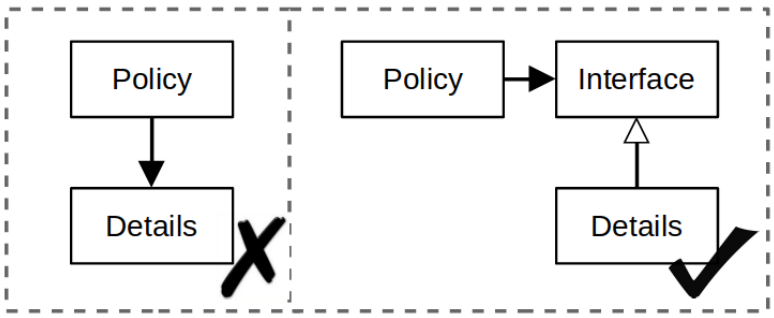

Figure 5.2: Dependency Inversion Principle. Left-hand side has high-level policy depending on low-level details, which is not recommended. Right-hand side has the dependency inverted by the policy depending on some interface, which the details extend.

other. Instead, an *abstract definition* of what the module should provide is used. This principle and best practice allows high-level policies to be left untouched as low-level details are swapped or undergo change, which in turn makes reuse of the high-level principles as libraries possible. The concept of interface is thus central to the application of this principle and best practice as further discussed in **Section 5.2.4**. In Logtalk, interfaces are represented using *protocols*, but Prolog module systems do not provide an equivalent feature[1].

The dependence on an interface, shown in **Fig 5.2**, also differentiates the technique from dependency injection or meta-programming where the context of the low-level details are passed to the high-level policy. With or without meta-programming, the policy is dependent on the predicates required being present in the details, however with meta-programming the interface is implicit, ungoverned, and not self-documenting. By using a declared interface as a first-class language feature the required predicates become explicit, the details are governed through a declared promise to define the interface, implementers of the interface can be enumerated using language reflection predicates, and documentation can be automatically generated.

These principles are also considered at the component level (Martin 2018). At this level of abstraction, the key ideas are to enable *reuse* through sensible component contents and *decoupling* through dependency cycle elimination as well as correlating dependency with stability.

5.2.2 Whitby before refactoring: Pre-Whitby

The first version of Whitby, written in Prolog using the module system and depicted in **Fig 5.3**, shall be called *Pre-Whitby* to distinguish it from Whitby after the refactoring. This application

[1]The ISO Prolog standard for modules (ISO/IEC 2000) does specify a module interface language construct but only allows a single implementation per interface, thus defeating the main purpose of defining interfaces. Moreover, this standard is ignored by Prolog systems.

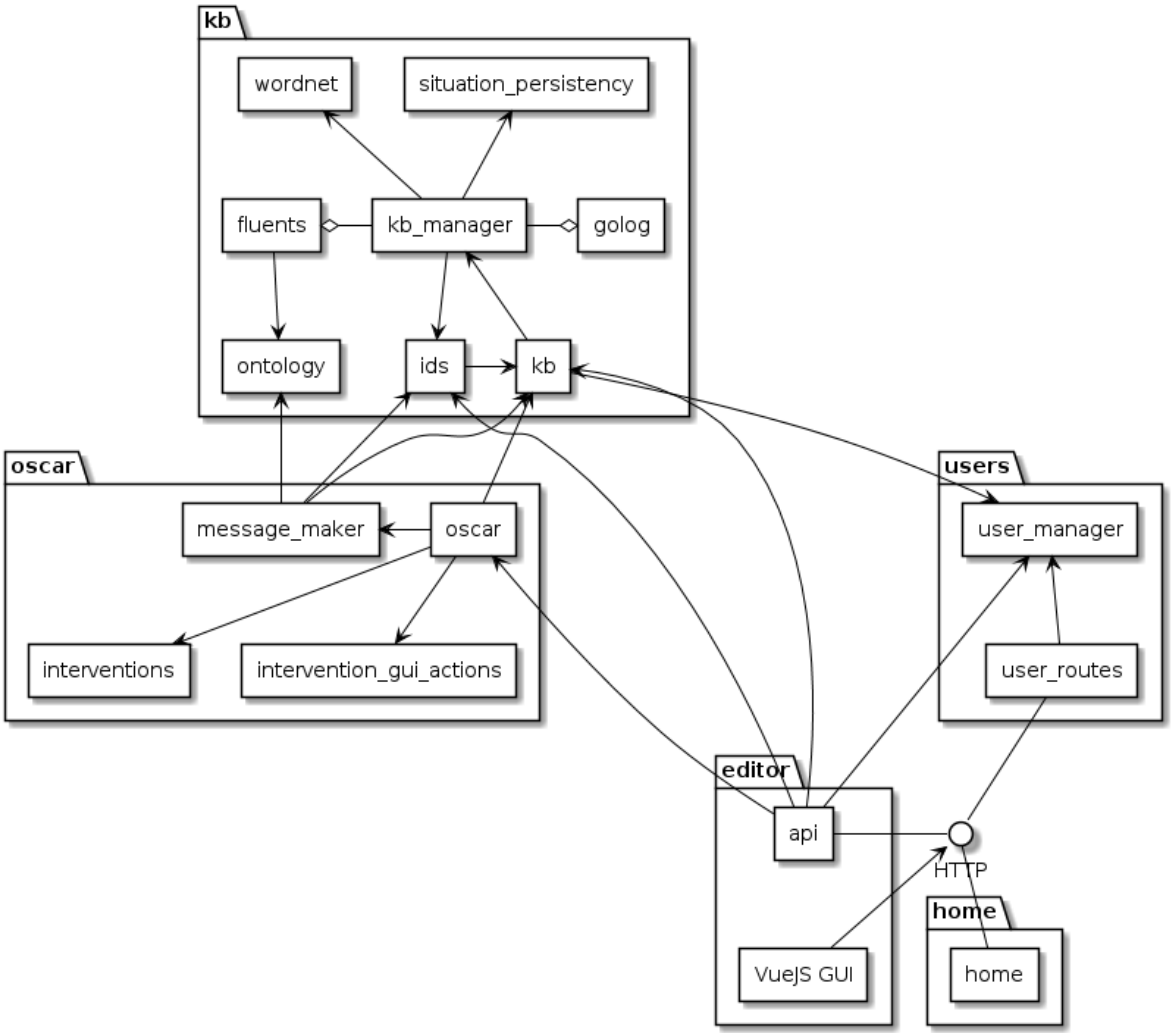

Figure 5.3: Dependencies in Pre-Whitby. Each node is a file, within their directories, which distinguish modules. Arrows denote imports, open diamonds denote consults.

did implement the features required of it[2]. There was no particular issue with it from a *user* perspective. The issues were entirely at the *developer* experience level.

The `kb` directory in **Fig 5.3** is responsible for the ontology authoring with situation calculus. The `oscar` directory, which is responsible for the contingent scaffolding interventions, is only dependent upon the code within this module. This dependence is a necessity as `oscar` needs to know about a user's ontology in order to offer relevant interventions.

Several violations of the SOLID principles are hidden at this level of abstraction. Note that `golog` and `fluents` are aggregated in `kb_manager`. This compromise is necessary because the Golog Situation Calculus reasoner from Reiter (2001) includes a predicate that calls these fluents, and so `golog` is dependent upon `fluents`. It is not uncommon in Prolog to apply some set of rules, like those in `golog`, to some facts, like those in `fluents`.

But for `golog` to reason over the `fluents` and `actions` defined for some particular world under analysis requires that world (details) to be imported into `golog` (policy). In this manner, the abstract is dependent upon the concrete, the stable is dependent upon the flexible, the calculus is dependent upon its own application. It's a violation of clean architecture that prevents code reuse: `golog` cannot be extended to include a defined world to reason over without modification to its own source code. Concurrent handling of multiple defined worlds is also precluded. Although there are workarounds to these problems, some of which are discussed here, they are unsatisfactory as the lack of necessary language constructs to cleanly express the application architecture results in the violation of SOLID principles.

To circumvent the issue of circular dependencies in Pre-Whitby, the `golog` and `fluents` files were consulted instead. This loaded them both into the `kb_manager` namespace. However, this causes a conundrum to resolve as there are fluents and actions for the `oscar` module that need to be defined in the `fluents` and `kb_manager` file so that `golog` is in the same namespace as them. For example, actions pertaining to ontology authoring, contingent scaffolding, and user interface are defined adjacent to each other in `kb_manager`, in violation of the Single Responsibility Principle, as seen in this snippet:

```
:- consult(kb(golog)).
:- consult(kb(fluents)).
```

[2]The only feature requirement that changed between Pre-Whitby and Whitby was a change from handling users with logins and many projects, to only handling many projects. Therefore this aspect is not compared.

```
%! action(Action, GologPossQuery)

% Ontology Authoring
action( add_data(_User, _Time, Payload),
        -asserted(Payload)).
action( delete_data(_User, _Time, Payload),
        asserted(Payload)).

% User Scaffolding Actions
action( dismiss_intervention(_User, _Time, Fact, Level),
        intervened(_, Fact, Level, _)).
action( request_intervention_increase(_User, _T, ID, Fact, Level),
        intervened(ID, Fact, Level, _)).

% Agent Scaffolding Actions
action( intervene(_User, _Time, Fact, Level, _Payload),
        -some(n, (dismissed(Fact, n) & n >= L))).

% User UI Actions
action( navigate_to_step(_User, _Time, _Step), true).
action( concept_focus(_User, _Time, _Focus), true).
action( glossary_lookup(_User, _Time, term(_Term)), true).
action( nudge(_User, _Time, _R), true).
```

Code belonging to `oscar` (One of the directories in **Figure 5.3**) resides in files in `kb`, which is then loaded into a different module. It should reside in files in `oscar` and somehow be made visible to `kb` to maintain separation of responsibilities and to ease code navigation. There are mechanisms to achieve this in Prolog: via consulting which would warn if a predicate were redefined, or via `include/1`, which includes the text of the file within the other.

To use `include/1` directives or multifile predicates would be to take code from `oscar` and have it effect the behaviour of `kb`; thus a developer working on either module must understand how

the other one is working. A poorly placed cut, unfortunately named predicate, or redefinition of an operator in `oscar` could cause `kb_manager`, upon which it depends, to no-longer function correctly. It opens up the potential for the consumer of some code to break what it should only depend upon. A developer debugging `kb` looking solely at their code in `kb`, believing it has no dependencies as the architecture diagram shows, would have little hope of resolving such an error. For these reasons module systems are favoured over the older `consult/1` and `include/1` predicates and why using them is also an unsatisfactory solution. When authoring Pre-Whitby, the more robust unsatisfactory solution was chosen, putting `oscar` code into `kb`, violating the Open-Closed principle and preventing code reuse, but easing debugging.

Another mechanism tried for including actions from different modules into `kb_manager` was to declare `action/2` as a multifile predicate in `kb_manager`. Clauses for the predicate could then be defined in `oscar` and any other module by using a prefix: `kb_manager:action(...)`. However, this violates the Dependency Inversion Principle as high-level policy predicates belonging to general ontology authoring and scaffolding are then dependent on the low-level detail that is the `kb_manager`, which is the module responsible for updating and querying user projects extending the ontology. Furthermore, this prefix referring to a specific, fixed module would prohibit the substitution of that module, thus violating the Liskov Substitution Principle of SOLID. Together this prevents the code in `oscar` from being used with a knowledge base that is not named `kb_manager` and using the same workaround.

Ideally a module would be used but the dependency needs to be inverted, such that `fluents` depend upon `golog`. Another workaround within the module system would be to make `kb_manager` dependent upon `fluents` and `golog` directly. Then the module can be included with the fluent or action where it is defined in all queries to Golog[3]. For example:

```
holds(Module:Fluent0, Situation) :-
  Module:restoreSitArg(Fluent0, Situation, Fluent),
  Module:Fluent.
holds(Module:Fact, Situation) :-
  not Module:restoreSitArg(Fact, Situation, _),
  isAtom(Fact),
  Module:Fact.
```

[3]Available at: http://www.cs.toronto.edu/cogrobo/kia/

This example also represents the resultant code of one strategy attempted via using meta-predicates to invert the dependency without using the interface depicted in **Fig 5.2**. With `holds/2` defined as a meta-predicate the *calling context is passed implicitly*, but `restoreSitArg/3` will be defined in the same module as the `Fluent`, which may be in a different module from the calling context: in Whitby there are multiple calling contexts, whereas each fluent is defined once. To make `restoreSitArg/3` available to the calling context would result in name-clashes when more than one module defining fluents is used. Therefore the dependency module where the definitions reside needed to be passed (or injected) for context as per this example.

The concern for this example is in the Golog call to `Module:Fluent`, where `Fluent` could be anything, including a meta-predicate, given in the query, which is a qualified call potentially breaking the encapsulation of the module.

Furthermore, it's no longer possible to use `holds/2` with a variable as the first argument to find fluents that hold in a ground situation without explicitly enumerating all modules and testing if they define fluents or not; the lack of protocols/interfaces as first-class entities precludes a simple and clean enumeration of only those modules that would declare conformance to a given protocol. The import semantics of Prolog modules also would force the use of these explicitly-qualified calls for the conforming modules to prevent predicate import clashes. This goes against what is considered best practice with Prolog modules: the use of implicit imports and implicit module-qualified predicate calls. But that is not the primary issue: by making a module that defines fluents and actions an *explicit* argument, we are forced to anticipate all predicates that, although not accessing fluents and actions directly, may be indirectly calling a predicate that requires that access (and thus require the module argument to be passed from upstream).

In Whitby however, which makes use of the required language constructs provided by Logtalk, `SitCalc` is loaded as a third-party package. As Logtalk does not use module-like imports semantics, there are never any loading conflicts when two or more loaded objects define the same public predicates. Furthermore, Whitby also loads packages defining ontology authoring terms and contingent scaffolding terms. The only place in Whitby where the contents of those packages need be considered is in the use of their fluents in queries of a situation and in the doing of their provided actions, both of which are done without the requirement to explicitly define the correct context to reason about them in.

Although dependency inversion is the crucial issue, there are additional violations of clean ar-

chitecture that need to be addressed. The dependency cycle between `kb_manager`, `ids`, and `kb` can cause a small edit in one of them to have perpetual ramifications as its dependency graph is also adapted to the change. Golog is more than a Situation Calculus reasoner; it is a parser for a Situation Calculus based language; thus `kb` is depending on code that it does not use. Furthermore, the four dependencies from the `oscar` module to the `kb` module suggest substitution would require more effort than should be necessary.

5.2.3 Refactored Whitby

The abstract architecture of Whitby is depicted in **Fig 5.1**, whereas a detailed view is in **Fig 5.4**. From the abstract view it can be seen how Whitby was designed to decouple the components of Pre-Whitby enabling code reuse. It is not possible to layer Pre-Whitby in a similar manner as in **Fig 5.1** due to the compromises made and tight-coupling.

As shown in **Fig 5.1**, ''`SitCalc` provides the theoretical foundation, which can be used to tackle a multitude of problems, it depends on nothing. The next "Highly Reusable Domain Layer" is the application of `SitCalc` to two domains; these libraries depend on `SitCalc`, but nothing in Whitby. Therefore they can be reused by any application wishing to apply Situation Calculus to Contingent Scaffolding or Ontology Authoring. The "Application Layer" is the core of the Whitby application, it is this code that applies the reusable libraries to the particular task at hand: Contingent Scaffolding an STPA analyst who is unwittingly authoring an extension to an ontology. Finally the "Interface Layer" provides a convenient means for the user to interact with the application.

The architecture, shown in **Fig 5.4**, initially appears more complex than Pre-Whitby in **Fig 5.3** as the third-party libraries that were extracted are also included, together with the protocols used to achieve dependency inversion. **Fig 5.5** shows only the internal dependencies: how the application appears to a developer working on it. Such a developer need not concern themselves with the working of any of the imported libraries; they are only responsible for what is depicted in **Fig 5.5**, a much simpler view.

From the developers perspective adding behaviour is also relatively simple. Whitby required a fluent describing what the user is looking at in the GUI; this is particular to the application of Whitby and so is not defined in `OntAuth`. To add this fluent to Whitby requires creating a new object that conforms to the `fluent_protocol`: new behaviour via extension rather than

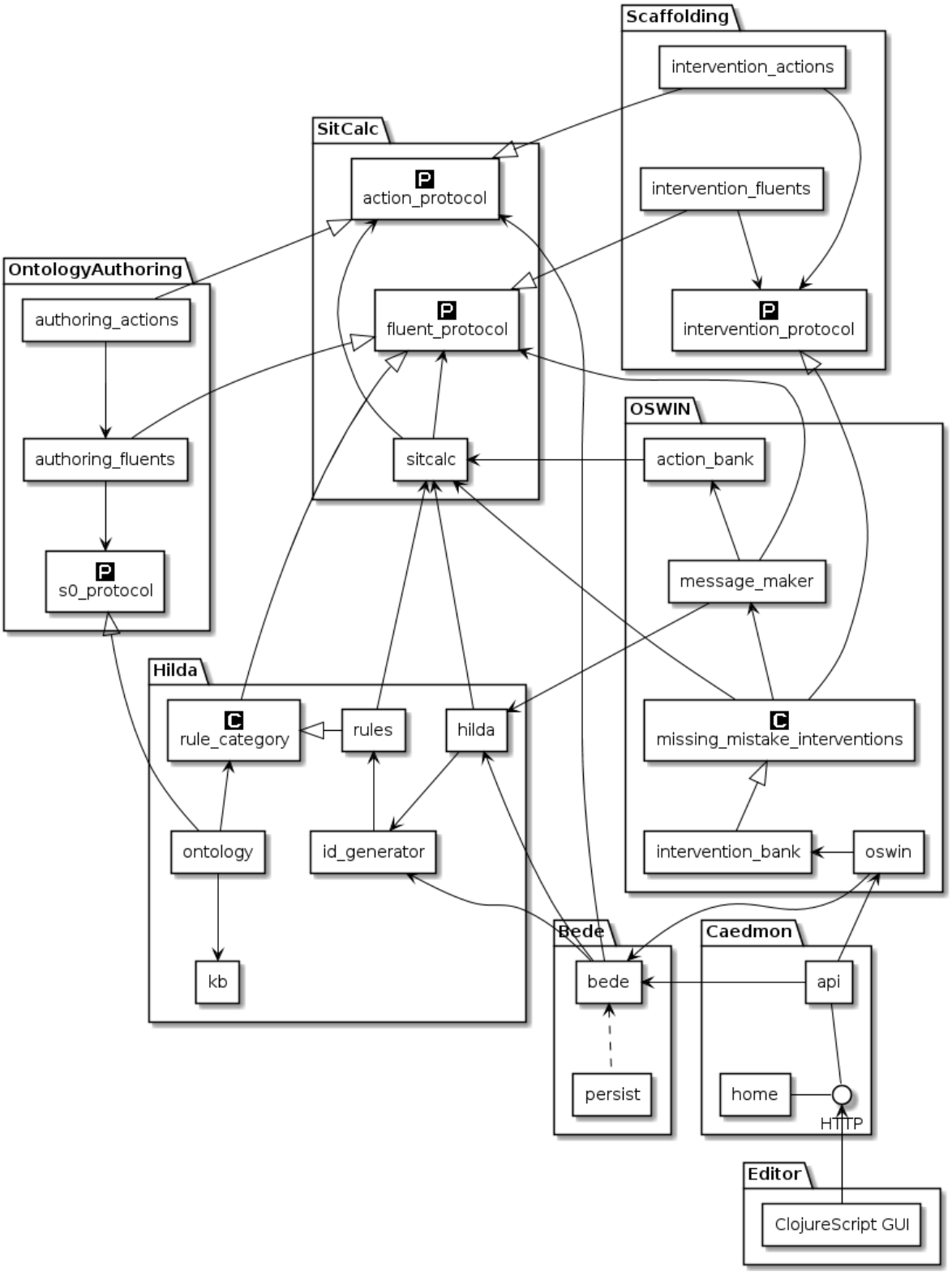

Figure 5.4: Dependencies in Whitby and extracted libraries. Each node (without a mark) is an object, within their directories. Protocols are marked with a "P", categories with a "C". Closed arrows denote dependence, open arrows denote implementation or extension, dashed arrow denotes event monitoring. `Bede` persists the Situation Calculus (`SitCalc`) logs and `Hilda` queries them for both frameworks: ontology authoring (in library `OntologyAuthoring`) and Contingent Scaffolding (in library `Scaffolding`). `OSWIN` applies the Contingent Scaffolding framework to this STPA domain.

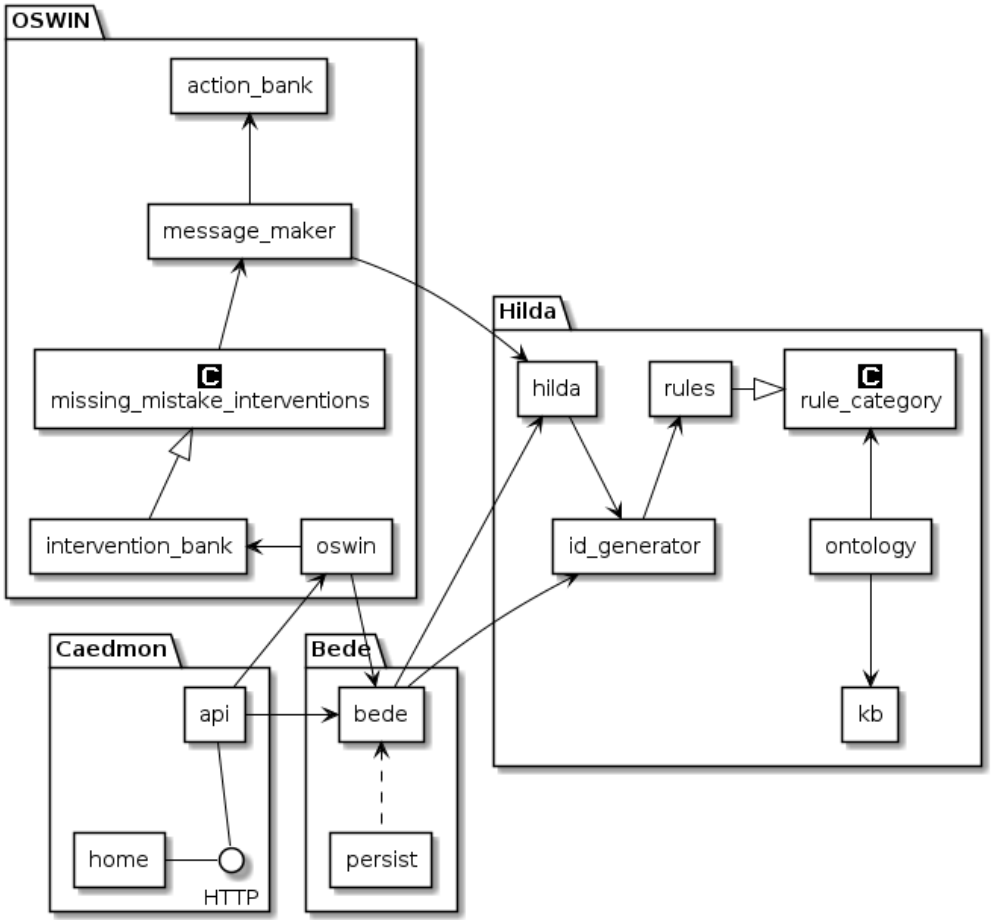

Figure 5.5: Dependencies within Whitby only. Each node is an object, within their directories. Arrows denote dependence, open arrows denote implementation or extension, dashed arrow denotes event monitoring. Categories are marked with a "C".

modification and exposing predicates that the application developer has no business editing, unlike prior to the refactor.

Fig 5.5 is a cleaner architecture, with no dependency cycles. However it is not yet perfect. For example, `bede` should not depend upon `id_generator`. That particular predicate should be exposed through `hilda` to provide an interface enabling easier substitution. Early in the refactoring to Whitby, each of the named directories was implemented as its own microservice communicating over HTTP. Correcting this issue would make it simple to split Whitby back into microservices for scalability, which isn't possible to achieve with Pre-Whitby due to the tight coupling between components.

5.2.4 Dependency Inversion using Logtalk Protocols

To achieve the desired architecture requires the application of the Dependency Inversion Principle, which can be accomplished via the Abstract Factory design pattern (Gamma et al. 1997; Martin 2018) described as:

> "Provide an interface for creating families of related or dependent objects without specifying their concrete classes." (Gamma et al. 1997)

In logic programming, we can reinterpret the implicitly imperative idea of *creating families* as *declaratively defining families*. Therefore, with Logtalk it becomes possible to do Dependency Inversion without dynamically creating objects. The concept of interface, in turn, is readily available using Logtalk *protocols*, as described below.

The Dependency Inversion Principle is applied to decouple the application into three major components. First a `SitCalc` library is extracted. Then `SitCalc` is extended, not modified, to create `OntAuth` and `Scaffolding` libraries. Finally, Whitby is created by importing these libraries as third-party libraries. The final architecture, with these libraries included, is shown in **Fig 5.4**. We start with a brief overview of Logtalk followed by a detailed account of how we applied this design principle to each component.

Logtalk Overview

Logtalk as a language reinterprets object-oriented concepts from first principles to provide logic programming with code encapsulation and code reuse mechanisms that are key in expressing well understood design principles and patterns (described in depth in "The Logtalk Handbook' '

(Moura 2021)). A key feature is the clear distinction between *predicate declarations* and *predicate definitions*[4], which can be encapsulated and reused as follows:

- **protocols**: Group *functionally cohesive* predicate declarations that can then be implemented by any number of *objects* and *categories*. Allows an object or category to promise conformance to an interface.
- **objects**: Group predicate declarations and predicate definitions. Objects can be *standalone* or part of hierarchies. Object enforce encapsulation, preventing calling predicates that are not within scope. Predicates are called using *message sending*, which decouples calling a predicate from the predicate definition that is used to answer the message.
- **categories**: Group a *functionally cohesive* set of predicate declarations and predicate definitions, providing a fine-grained unit of code reuse that can be imported by any number of objects, thus providing a *composition* mechanism as an alternative to the use of inheritance.

Predicates can be declared *public*, *protected*, or *private*. A predicate declaration does not require that the predicate is also defined. Being able to *declare* a predicate, independent of any other predicate properties, without necessarily *defining* it, is a fundamental requirement for the definition of protocols. It also provides clear *closed world semantics* where calling a declared predicate that is not defined simply fails instead of generating an error (orthogonal to the predicate being *static* or *dynamic*).

Logtalk defines a comprehensive set of *reflection* predicates for reasoning about the use of these components in the program. In particular, the `conforms_to_protocol/2`, which is true if the first argument implements or is an extension of something that implements the protocol named in the second argument, and `current_object/1`, which is true if its argument is an object in the application current state (categories and protocols have their own counterparts). These predicates are used in the implementation of the SOLID principles as illustrated in the next sections.

Logtalk also provides a comprehensive set of portable developer tools, notably for documenting, diagramming, and testing that were used extensively. These tools reflect how the language constructs are used in applications, from API documentation to diagrams at multiple levels of

[4]This distinction exists in standard Prolog (ISO/IEC 1995) only for predicates declared as *dynamic* or *multifile*. Notably, *static* predicates exported by a module must be defined by the module.

abstraction that help developers and maintainers navigate and understand the code base and its architecture.

5.3 Reusable Libraries

This section describes the application of SOLID with Logtalk to author the `SitCalc`, `OntAuth`, and `Scaffolding` libraries.

5.3.1 A Reusable SitCalc Library

The SitCalc libraries[5] can be used for reasoning with Situation Calculus. It includes predicates that need to send messages to fluent and action objects. Rather than depend on these fluents and actions directly, it depends instead on objects conforming to `action_protocol` and `fluent_protocol`. This is the Dependency Inversion Principle of SOLID: to depend only on protocols/interfaces and not on concrete code (Martin 2018). These protocols declare the predicates that an action and fluent are expected to define:

```
:- protocol(action_protocol).

:- public(do/2).
:- info(do/2, [
  comment is 'True if doing action in ``S1`` results in ``S2``.',
  argnames is ['S1', 'S2']
]).

:- public(poss/1).
:- info(poss/1, [
  comment is 'True if the action is possible in the situation.',
  argnames is ['Situation']
]).
:- end_protocol.
```

[5]The Situation Calculus libraries, including SitCalc, are made publicly available at `https://github.com/PaulBrownMagic/woolpack` and can be installed with the Logtalk packs tool.

```
:- protocol(fluent_protocol).

:- public(holds/1).
:- info(holds/1, [
  comment is 'True if the fluent holds in the situation.',
  argnames is ['Situation']
]).
:- end_protocol.
```

Thus any object that is an action or fluent can be found or validated, using the Logtalk built-in reflection predicates[6]. For some strategies attempted without an interface in Pre-Whitby, such as when passing the definition context explicitly (as previously discussed in **Section 5.2.2**), the enumeration of modules requires a hand-coded alternative to mark the modules, which is fragile and not self-documenting. These predicates are used to validate or enumerate:

```
is_action(Action) :-
  conforms_to_protocol(Action, action_protocol),
  current_object(Action).

is_fluent(Fluent) :-
  conforms_to_protocol(Fluent, fluent_protocol),
  current_object(Fluent).
```

Now within the `sitcalc` object when it is necessary to call a fluent or action they can be called, even if the argument is a variable, without depending on the fluents or actions. Here are two extractions from the code within `sitcalc` that demonstrate doing so:

```
holds(Fluent, Situation) :-
  is_fluent(Fluent),
  Fluent::holds(Situation).
```

[6]The `conforms_to_protocol/2` predicate enumerates both objects and categories that implement a protocol. As we are only interested in objects, we use the `current_object/1` predicate to filter out any categories as these are used only to provide common definitions for utility predicates.

```
poss(Action, Situation) :-
    is_action(Action),
    Action::poss(Situation).
```

In addition to this, the extraneous code in Golog is not included in `SitCalc` such that unused code is not depended upon.

There is more than one way to represent a situation in Situation Calculus: either as a history of actions or as a collection of fluents. The first method, as used by this application, is to keep the situation term as Reiter does (Reiter 2001), as a log of the history of actions. This method can reason over part situations and the actions taken to reach any particular situation from the initial one. However, if that log of actions becomes long then reasoning with context dependent fluents can become slow. This method is implemented as the SitCalc library.

The alternative method is to store the situation as only a collection of the holding fluents, adding and removing fluents as they become true or false respectively. This method is similar to the classic STRIPS planner and is also discussed by Reiter (2001). With this method the history of actions is lost but the computational complexity of checking if any fluent in the situation holds is a function of the number of fluents that currently hold in the situation, rather than the number of actions taken so far. This method is implemented as the STRIPState library, which is not used in this application but only provided and described for reuse and completeness.

Both SitCalc and STRIPState share a common interface to ease learning. Within Logtalk they're defined using protocols in the Situation library, upon which both SitCalc and STRIPState depend. Therefore, objects that are fluents promise a `holds/1` predicate and actions promise `do/2` and `poss/2` predicates. For convenience, and to introduce more features, a situation category is also provided that includes ways to iterate over all loaded actions and fluents, as well as an interface for compound queries regarding what fluents hold in a situation, inspired by Golog (Reiter 2001). Thus rather than querying like so:

```
?- temperature(water, WT)::holds(S),
     temperature(milk, MT)::holds(S),
     \+ MT > WT.
```

This can be written in a syntax that's easier for a non-Prolog programmer to write, in an effort to make it more accessible to domain experts. So a non-Prolog programmer can write the

expression unified with `Query`, which the Prolog programmer can setup the query interface to `sitcalc` for:

```
?- Query = temperature(water, WT) and
   temperature(milk, MT) and not MT > WT,
   sitcalc::holds(Query, S).
```

This domain specific query language for fluent queries extends the built in Prolog operators for conjunction, disjunction and its "negation as failure" compromise of negation, with an implication and equivalence operators. These are also replaced with words to eliminate the need for a non-specialist user to learn the symbols:

```
:- op(800, xfy, and).
:- op(850, xfy, or).
:- op(870, xfy, implies).
:- op(880, xfy, equivalentTo).
:- op(600, fy, not).
```

However due to Prolog's special kind of negation, the following caveats apply:

- For the case of `not P`, Prolog style negation is used (\+)
- For the case where `P implies Q`, `P` is false, and `Q` is not ground, `Q` can't be unified with a false case
- For the case where `P equivalentTo Q` if either `P` or `Q` are not ground only the case where both `P` and `Q` are true will be unified. If either is ground to a false case and the other is not ground, it cannot be unified with a false case which would make `P equivalentTo Q` true.

This domain specific query language inherits the same defaulty representation of Prolog queries (O'Keefe 1990). The defaulty representation occurs when unifying terms in multiple heads for the same functor when a catch-all head is required. This means whenever a unification match is made with another head an unnecessary choice point is added because that term will also match the catch-all head. As this code is core to situation calculus applications the performance of this predicate is especially important, and so it's important to circumvent this issue. In the case of this DSL implementation this occurs when checking if the query term is a compound query and not a variable:

```
compound_nonvar(not _).
compound_nonvar(_ and _).
compound_nonvar(_ or _).
compound_nonvar(_ implies _).
compound_nonvar(_ equivalentTo _).
compound_nonvar(Term) :-
    nonvar(Term).
```

The method of circumvention used, while particular to this case and something of a trick, is effective although not generalisable to all defaulty representations. As a variable is not allowed at all, an error can be thrown if one is found, which will remove further choice points for following heads. Therefore, a term that is not expected to be used as a fluent is placed in the head that only a variable (or a fluent of the same name with arity 0), will unify with it:

```
% unification trick to test for variable
compound_nonvar('Variable Fluents are not supported, at least some
    part of the Fluent term must be ground') :-
    context(Context),
    throw(error(instantiation_error, Context)).
% test if arg is a query term that requires transformation
compound_nonvar(not _).
compound_nonvar(_ and _).
compound_nonvar(_ or _).
compound_nonvar(_ implies _).
compound_nonvar(_ equivalentTo _).
```

The obvious flaw in this solution is that it precludes one possible fluent name from the library user. However, it's possible to choose an atom that's so improbable for a user to choose for a fluent that it becomes negligible. The alternative solution would be to use a wrapper functor to identify each non-compound fluent clause. This is rejected primarily as it's an unnecessary burden on a non-expert programmer writing queries, and secondarily following the advice of O'Keefe (1990), who also warns against using the defaulty representation as well as unnecessary wrappers, both with regards to unnecessary performance costs. With `compound_nonvar/1` de-

fined the implementation of the query language, using the same Lloyd-Topper transformation as Golog (Reiter 2001), is done like so:

```
% query/2 is non-tabled holds/2
% for when tabling is supported,
% it reduces overhead of tabling
% each subquery
query(Query, Situation) :-
    ( compound_nonvar(Query) % check if fluent, compound, or var
    -> compound_query(Query, Situation) % decompose query
    ; ::holds_(Query, Situation) % check fluent
    ).
compound_query(not Query, Situation) :-
    \+ query(Query, Situation).
compound_query(P and Q, Situation) :-
    query(P, Situation),
    query(Q, Situation).
compound_query(P or _Q, Situation) :-
    query(P, Situation).    % Decompose query
compound_query(_P or Q, Situation) :-
    query(Q, Situation).    % Decompose query
compound_query(P implies Q, Situation) :-
    (    query(P, Situation)
    ->  query(Q, Situation)
    ;    ignore(query(Q, Situation))
    ).
compound_query(P equivalentTo Q, Situation) :-
    (    query(P, Situation)
    ->  query(Q, Situation) % P holds, so Q must also hold
    ;    \+ query(Q, Situation) % not P holds, so not Q must hold
    ).
```

The final feature added is memoization for these queries. When running queries with context

dependent fluents over situations with hundreds of actions, it can be that it takes seconds to compute. At the cost of memory this can be mitigated by using Prolog's tabled resolution for those Prolog dialects that provide it:

```
:- if(current_logtalk_flag(tabling, supported)).
        :- table(holds/2).
  :- endif
  holds(Query, Situation) :-
      % holds/2 might be tabled,
      % pass to query/2 to avoid overhead of tabling subqueries
      query(Query, Situation).
```

The same tabling strategy is used for individual fluents in the SitCalc library. It means that whenever the `holds/2` predicate is called with a `Situation` term it has seen before then it will get the result from memory rather than re-calculating it. When encountering a `Situation` term it hasn't seen before the last situation that was seen effectively becomes `s0`, avoiding the need to walk the whole `Situation` term on each query.

5.3.2 Extending SitCalc with Reusable Libraries

The two `OntAuth` and `Scaffolding` libraries both extend `SitCalc`, but both are also defined in a way that they can be used as third-party libraries with `SitCalc` as a dependency. They extend `SitCalc` by defining fluents and actions that are pertinent. `OntAuth` includes a fluent to see what triples hold in the initial situation. Here, `s0` is a *marker protocol*, allowing easy enumeration of initial situations by using the reflection predicates, and also dependency inversion via a protocol (`fluent` is a category that implements `fluent_protocol`):

```
:- object(initial_assertion(_Subject_, _Predicate_, _Object_),
  imports(fluent)).

  holds(_AnySit) :-
    conforms_to_protocol(S0, s0),
    current_object(S0),
    S0::asserted(_Subject_, _Predicate_, _Object_).
```

```
:- end_object.
```

`Scaffolding` includes an action to intervene (here `action` is a category that implements `action_protocol`):

```
:- object(intervene(_Intervention_, _Query_, _Lvl_, _Time_),
  imports(action)).

  poss(Sit) :-
    conforms_to_protocol(Interventions, interventions),
    current_object(Interventions),
    Interventions::intervention(_Intervention_, _Query_),
    sitcalc::holds(_Query_, Sit),
    intervention_level(_Intervention_, _Query_, _Lvl_)::holds(Sit),
    \+ live_intervention(_Intervention_, _Query_, _Lvl_)::holds(Sit).

:- end_object.
```

Both objects are *parametric* objects (Moura 2011). The object parameters (e.g. `_Subject_` are logic variables shared with all the object predicates.

Due to the implementation of `SitCalc`, all these fluents and actions are visible to `sitcalc` whilst it does not depend on them. However, these two examples both depend upon some implementation details: some `S0::asserted/3` and some `interventions::intervention/2`. These dependency issues are solved in the same manner as for `SitCalc`: through dependency on a protocol (as illustrated in **Fig 5.4**).

Between these two libraries a total of 14 fluent and action terms are introduced that can be queried via `SitCalc`. Although these depend on `SitCalc`, they do not depend on any application that makes use of them. Whitby is such an application, by importing these libraries it gains these 14 fluents and actions, needing only to implement both the `s0_protocol` and `intervention_protocol`. In contrast to Pre-Whitby, the contingent scaffolding is also not dependent on code that includes ontology authoring, meaning it can be applied to other activities than ontology authoring.

5.3.3 OntAuth Library

The ontology authoring library is intended to define all the actions and fluents necessary for authoring and reasoning with an ontology with SitCalc. As SitCalc is written in Logtalk the reasoning is done with Prolog, and so there are differences from the usual Description Logic reasoners used with OWL ontologies.

Ontology Authoring Actions

These align with the typical database type of actions, a user can:

- `assert(Triple, User, Time)`: that is to add some new subject-predicate-object triple to the ontology. It's possible when the triple is not already asserted.
- `retract(Triple, User, Time)`: that is to remove some subject-predicate-object triple to the ontology. It's possible when the triple is already asserted, including if it's in an initial ontology loaded in `s0`.
- `update(OldTriple, NewTriple, User, Time)`: that is to change some triple. It's possible when the two triples are different and if both `retract(OldTriple, User, Time)` and `assert(NewTriple, User, Time)` are possible. This provides an atomic way to do updates for multi-threaded applications.

In the implementation triples are stored as `spo(Subject, Predicate, Object)` terms.

These authoring actions import the `action` category from the SitCalc library, and so they are globally visible as actions without the application author needing to do anything more than load the library.

Ontology Authoring Reasoning

The reasoning is provided through fluents that can be used within SitCalc queries dependent on the application authors needs and with attention to the cost of reasoning.

At the root of every situation term is `s0`, and for this library to function with an `s0` that contains some pre-existing ontology, it needs to know the predicates to call to get that information. Therefore an `s0` protocol is provided that an application developer can implement with their own definitions, which is what is done in Whitby.

```
:- protocol(s0).
```

```
    :- public(asserted/3).
    :- info(asserted/3, [
         comment is 'A triple that is asserted to be true in s0',
         argnames is ['Subject', 'Predicate', 'Object']
       ]).
:- end_protocol.
```

Thus, in Whitby where the ontology being loaded into s0 is defined in OWL, the SWI-Prolog RDF libraries are used to load the triples. Terms are then stripped of the URI's and prefixes for developer convenience. The stripped terms are then asserted into `spo/3` predicates, which are held in a `kb` object as public predicates. So the definition of the s0 ontology object becomes as simple as:

```
:- object(stpa_ontology, implements(s0)).
     asserted(S, P, O) :-
         kb::spo(S, P, O).
:- end_object.
```

Furthermore, due to static binding, the message call to `kb::spo/3` is compiled away so there is no performance overhead here.

With this protocol the first reasoning fluent can be defined, which is to query what was asserted in `s0`, irregardless of any subsequent actions. This can provide a useful shortcut for access to facts that are known to be in the initial ontology if the users are prohibited by the application developer from retracting them. In Whitby it's used for tasks like retrieving the labels of STAMP ontology terms.

```
:- object(initial_assertion(_S_, _P_, _O_), imports(fluent)).
   holds(_Sit) :-
     implements_protocol(S0, s0),
     S0::asserted(_S_, _P_, _O_).
:- end_object.
```

The complementary fluent to `initial_assertion/3` is `logged_assertion/3`, which is a triple asserted by an action in the log, ignoring any assertions in `s0`. This is used in Whitby to

distinguish facts the user has asserted when considering if they will be permitted, or even encouraged, to change those facts.

```
:- object(logged_assertion(_S_, _P_, _O_), imports(fluent)).
   holds(do(assert(spo(_S_, _P_, _O_), _User, _Time), _Sit)).
   holds(do(update(_Old, spo(_S_, _P_, _O_), _User, _Time), _Sit)).
   holds(do(Action, Sit)) :-
      holds(Sit),
      \+ retraction(Action).

   retraction(retract(spo(_S_, _P_, _O_), _User, _Time)).
   retraction(update(spo(_S_, _P_, _O_), _New, _User, _Time)).
:- end_object.
```

These two assertion fluents are then combined into an `asserted/3` fluent that provides the means to query all fluents that have been asserted in a situation.

```
:- object(asserted(_S_, _P_, _O_), imports(fluent)).
     holds(Sit) :-
          logged_assertion(_S_, _P_, _O_)::holds(Sit).
     holds(_Sit) :-
          initial_assertion(_S_, _P_, _O_)::holds(s0).
:- end_object.
```

The "asserted" family of fluents provide ways to simply retrieve triples that have been explicitly asserted. Ontology, however, can be used to deduce additional information, which in this library is made available through a `fact/3` fluent (as defined in **Section 4.2**). This fluent supports inverse, reflexive, symmetric, and transitive relations, as well as sub-properties. However, due to time and efficiency constraints the implementation of `subClassOf` is treated only as a reflexive and transitive relation, meaning the reasoner cannot detect if one term is a subclass of another by analysis of the relations they hold but only through reasoning with the explicit `subClassOf` term. A correct subclass reasoner in Prolog was considered out-of-scope for this work and wasn't required for this application, therefore it has been left for further work.

5.3.4 Scaffolding Library

The scaffolding library is dependent on SitCalc and independent of OntAuth. It defines the necessary terms to implement Contingent Scaffolding following the framework as defined in **Section 4.3.2**. Note that the library does not use or define any of the intervention messages created or used, that functionality is left to the developer as it will be application and UI specific.

This library is dependent on some user-defined intervention terms being accessible. Following the same design-pattern used before, a protocol is provided for the application developer to implement such that the Scaffolding library can query the user-defined intervention terms:

```
:- protocol(interventions).
    :- public(intervention/2).
    :- mode(intervention(+atom, +term), zero_or_more).
    :- info(intervention/2,
        [ comment is 'An Intervention with associated query'
        , argnames is ['ID', 'Query']
        ]).
:- end_protocol.
```

An intervention is defined with some identity term, which also can be used to give a quickly human-readable label, and some query, which is a Situation Calculus query that when true means the intervention is possible to do.

Scaffolding Actions

There are two user actions and one software action to define. A user can `request_help/3` or `dismiss_intervention/3` for a live intervention, identified by the `live_intervention/3` fluent. The software action is to `intervene/4`, where the first two arguments correspond to the identity and query of the intervention, the third argument is the level the intervention is being done at with respect to the framework, and the last argument is for the time, but is unused by the Scaffolding library.

It's possible to do an intervention when the associated query holds, so that the message to `intervene/4::poss/1` can be called with variable terms in search for an intervention that's possible, the intervention protocol is used to iterate over all defined interventions, in the same way

as the Situation Calculus library does for `sitcalc::is_action/1` and `sitcalc::is_fluent/1`. If one is found where the query succeeds and that's not a current live intervention, `intervention_level/3` fluent is checked to see at what level the intervention should be done.

Scaffolding Fluents

There are two key fluents in the library: `live_intervention/3` and `intervention_level/3`. An intervention is live if the `intervene/4` action has been done, it's not been dismissed, and if the query that made it possible still succeeds and has succeeded in all the intermediate situations. This is an expensive computation to do without tabling, it's also pervasive, being required for every scaffolding action `poss/1` check and being very useful when prioritising interventions to present to a user in the application.

```
:- object(live_intervention(_Intervention_, _Query_, _Level_),
    imports(fluent)).

    holds(Sit) :-
        holds(Sit, Sit).

    holds(do(intervene(_Intervention_, _Query_, _Level_, _Time),
                       _SitTail), CurSit) :-
        sitcalc::holds(_Query_, CurSit).
    holds(do(A, SitTail), CurSit) :-
        A \= intervene(_Intervention_, _Query_, _Level_, _Time),
        A \= dismiss_intervention(_Intervention_, _User, _Time),
        holds(SitTail, CurSit), % grounds query, so check can be done
        sitcalc::holds(_Query_, do(A, SitTail)).

:- end_object.
```

An alternative to tabled resolution would be to monitor when an intervention no longer becomes live and to insert some action at this point to mark it in the situation term. This approach hasn't been followed as it's an artificial action that no party took and tabled resolution is sufficient. In this way the log of actions is still a log of actions and not contaminated with pseudo-actions

added for computation purposes only.

The `intervention_level/3` fluent follows from the formal definition (**Section 4.3.2**), taking advantage of Prolog's multiple clause syntax to aid readability.

```
:- object(intervention_level(_Intervention_, _Query_, _Level_),
    imports(fluent)).

    holds(Sit) :-
        once(holds_(_Query_, Sit)).

    % no-change
    holds_(Query,
           do(intervene(_Intervention_, Query, _Level_, _Time),
           _S)).

    % increment
    holds_(Query,
           do(request_help(_Intervention_, _User, _Time),
           S)) :-
        intervention_level(_Intervention_, Query, N)::holds(S),
        succ(N, Inc), _Level_ is min(Inc, 4).

    % reset
    holds_(_Query,
           do(dismiss_intervention(_Intervention_, _User, _Time),
           _S)) :- _Level_ = 1.

    % decrement
    holds_(Query, do(A, S)) :-
        sitcalc::holds(Query, S), % intervention was valid
        \+ sitcalc::holds(Query, do(A, S)), % but now resolved
        intervention_level(_Intervention_, Query, N)::holds(S),
```

```
        succ(Dec, N), _Level_ is max(Dec, 1). % so fade

    % recur, not barring choice points for decr for efficiency
    holds_(Query, do(A, S)) :-
        A \= intervene(_Intervention_, Query, _, _),
        A \= request_help(_Intervention_, _, _),
        A \= dismiss_intervention(_Intervention_, _, _),
        holds_(Query, S).

    % s0, it exists and default is 1 (0 is when it's not possible)
    holds_(_Query, s0) :-
        _Level_ = 1.

:- end_object.
```

This fluent is highly context dependent, with many calls to other fluents. Again, tabled resolution is used to mitigate the computation time, reducing the cost of a `holds/2` call to `O(1)` on the second time it's called with the same variable bindings. The `once/1` call in `holds/1` is used to cut any remaining choice points as there is only one valid answer to what the intervention level is, but without cutting any choice points prior to this call that a user may have left in their application.

5.4 Contingent Scaffolding for STPA Implementation

With the generalised libraries defined for Situation Calculus, Ontology Authoring, and Contingent Scaffolding, they need to be applied to STPA within the application. This is done in the component named OSWIN, which given a situation can deduce any intervention to be proffered to the user, in a format agreeable to being sent to the UI as a payload. Therefore OSWIN uses a bank of interventions and message generating reasoning to determine the messages to send.

5.4.1 OSWIN Intervention Bank

The definition of an intervention requires an identity, so we can log what intervention was fired, and a query, so that we can determine if the intervention should be fired. It also needs to support

messages for all the levels of contingent scaffolding:

1. General prompt
2. Leading question
3. Specific instruction
4. Physical intervention

Message Generation

The general prompt is the most general message, it describes the intervention and denotes the kind of intervention offered. Therefore this is used for the identity and is included in the log. Whilst it's possible to generate the rest of the intervention messages from the query, it's not always advisable.

The leading question was originally translated from the query via a Prolog DCG, such that a query ground as:

```
logged_assertion('H-1', subClassOf, 'Hazard') and
  not (logged_assertion(Loss, subClassOf, 'Loss') and
  logged_assertion('H-1', hasPossible, Loss))
```

Would become: "Have you not asserted that 'H-1' is a kind of 'Hazard' and some 'Thing' is a kind of 'Loss' and that 'H-1' has possible 'Thing'?". These confusing messages become even harder for a human to parse as the number of clauses grow. Rather than burden the user with the task of trying to read clunky and grammatically incorrect English, it was decided to burden the one defining the interventions with the additional task of writing a leading question with variable substitutions, like so:

```
'Have you not asserted which Loss is possible to occur in
 the ~s Hazard?'-[Hazard]
```

Which for the same query grounding becomes the leading question: "Have you not asserted which Loss is possible to occur in the H-1 Hazard?".

The message generating DCG is still used for the instruction generation, without becoming unintelligible. In this intervention the message is narrowed down to a single action the user can take in order to change some succeeding clause in the query to fail. Such that the intervention

will not fire. This is somewhat counter-intuitive as the goal is to make the query fail rather than succeed.

The Lloyd-Topor transformations are used, as found in Golog (Reiter 2001), and the first succeeding sub-goal of the reversed and reduced query is used to generate the instruction. The query is reduced to remove terms that a user cannot change (such as $3 > 2$), in this case nothing is excluded by the reduction. The query is reversed so that the last clauses are instructed on first as they tend to be better candidates for editing, depending upon variables ground in earlier clauses. The reduced and reversed query becomes:

```
not (logged_assertion('H-1', hasPossible, Loss) and
        logged_assertion(Loss, subClassOf, 'Loss')) and
logged_assertion('H-1', subClassOf, 'Hazard')
```

This reduced query is then searched for its component fluents using the Lloyd-Topor transformations. This generates the single clauses one at a time of:

```
not logged_assertion('H-1', hasPossible, Loss) ;
not logged_assertion(Loss, subClassOf, 'Loss') ;
logged_assertion('H-1', subClassOf, 'Hazard') .
```

Each clause is considered individually to instruct the user on how to make that clause fail, whereas now it succeeds. The first clause begins with `not`, which indicates that the term should be failing, and indeed it does not hold in the situation that generated this semi-ground query (where 'H-1' has been defined without a `hasPossible` link to a loss). This agreement that it should not hold and does not hold is part of what is making this query succeed to fire the intervention. So the user is instructed to assert that some 'Loss' is possible in 'H-1'; the negation of the clause is passed through the Prolog DCG to create the instruction: "*You need to assert that 'H-1' has possible some 'Thing'.*"

The same can be followed for subsequent clauses if they continue to cause the intervention to fire, so if the user asserted the fact `spo('H-1',hasPossible,'L-1')` but not that `L-1` was a 'Loss', they'd next be instructed: "*You need to assert that 'Something' is a kind of 'Loss'*". The order of the clauses is important, as the final clause instruction would be: "*You need to not assert that 'H-1' is a kind of 'Hazard'*", which would also prevent the intervention from firing but likely moves the user away from their intended goal rather than aid them.

The physical intervention is generated in a similar manner. The last succeeding clause is found and reasoned with to create UI specific instructions that are sent to the UI to execute; generating ClojureScript rather than English.

Intervention Definition

Two kinds of interventions were determined[7]: potentially missing and a potential mistake. Mistake-type interventions can be immediately determined as to being required, such as when a user asserts something that is either provably wrong or they violate some advice. This intervention catches a logical contradiction: if the user asserts that some action both causes some fluent to hold and not hold in the subsequent situation.

```
mistake(
  'Control Action can\'t both cause a fluent to hold and not hold',
  logged_assertion(CA, subClassOf, 'ControlAction') and
    logged_assertion(CA, requestsToHold, F) and
    logged_assertion(CA, requestsToNotHold, F),
  'Have you asserted that the Control Action ~s requests ~F both holds
  and doesn\'t hold?'-[CA, F]
).
```

The missing kind of intervention can only be determined after a user has passed some point in the process, such as if a user were to go to step 2 without defining the losses required in step 1. This `missing/4` intervention is to catch the case where a user hasn't asserted that some hazard leads to some loss. This is an assertion that should be made during step 1, and this intervention will be checked if the user is in step 2.

```
missing(
  step1,
  'Hazards will lead to a loss in some worst-case environment',
  logged_assertion(Hazard, subClassOf, 'Hazard') and
    not (logged_assertion(Loss, subClassOf, 'Loss') and
    logged_assertion(Hazard, hasPossible, Loss)),
  'Have you not asserted which Loss is possible to occur in
```

[7] See **Section 4.3.2**

```
    the ~s Hazard?'-[Hazard]
).
```

These two kinds of interventions require different definitions and handling. A mistake intervention can be fired anytime it's possible. However, there is no need to even consider if a missing intervention pertaining to some step 2 action is possible when a user is still in step 1. Furthermore, the missing interventions provide a manner to advise a user on the minimal actions required for the current step: by simulating being at the next step the same reasoning code can be used to intervene on missing from the current step.

The complete list of defined missing and mistake interventions can be seen in **Appendix C**.

5.5 Tour of the Graphical User Interface

This section describes the design and capabilities of the interface part of the "Editor" component of Whitby, which is provided to the analyst in order to interact with the software.

5.5.1 Step 1 Interface

When a user begins their STPA analysis they are greeted by the screen in **Figure 5.6**. The STPA analysis interface contains two navigation bars, the uppermost one pertains to the user-study only. The second one provides a way for the user to navigate back-and-forth between the three steps they are to undertake. They are currently on step 1, therefore that navigation item is emphasized and they're presented with an interface for defining losses and hazards, which will be recorded by Bede so that Hilda can reason with them.

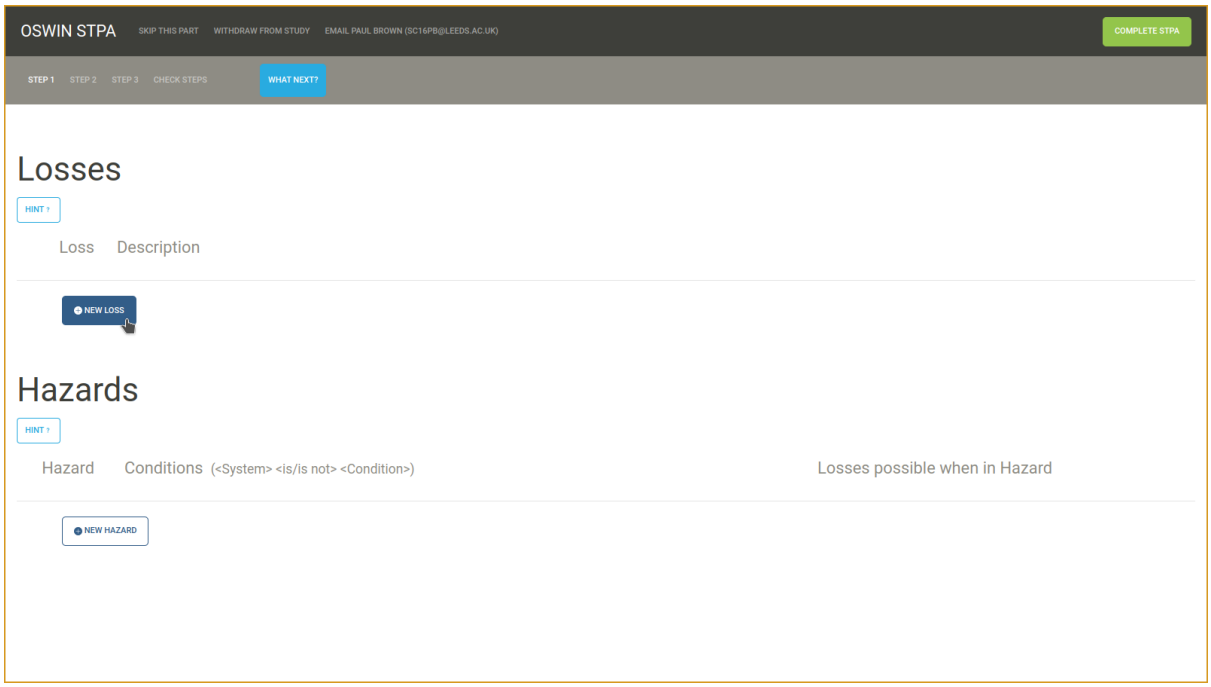

Figure 5.6: The editor interface is presented on step 1 with navigation to all 3 steps and the capability to define losses and hazards.

To add a loss the button highlighted in **Figure 5.6** is clicked. Losses are simple terms with only a label. Next is to add a hazard.

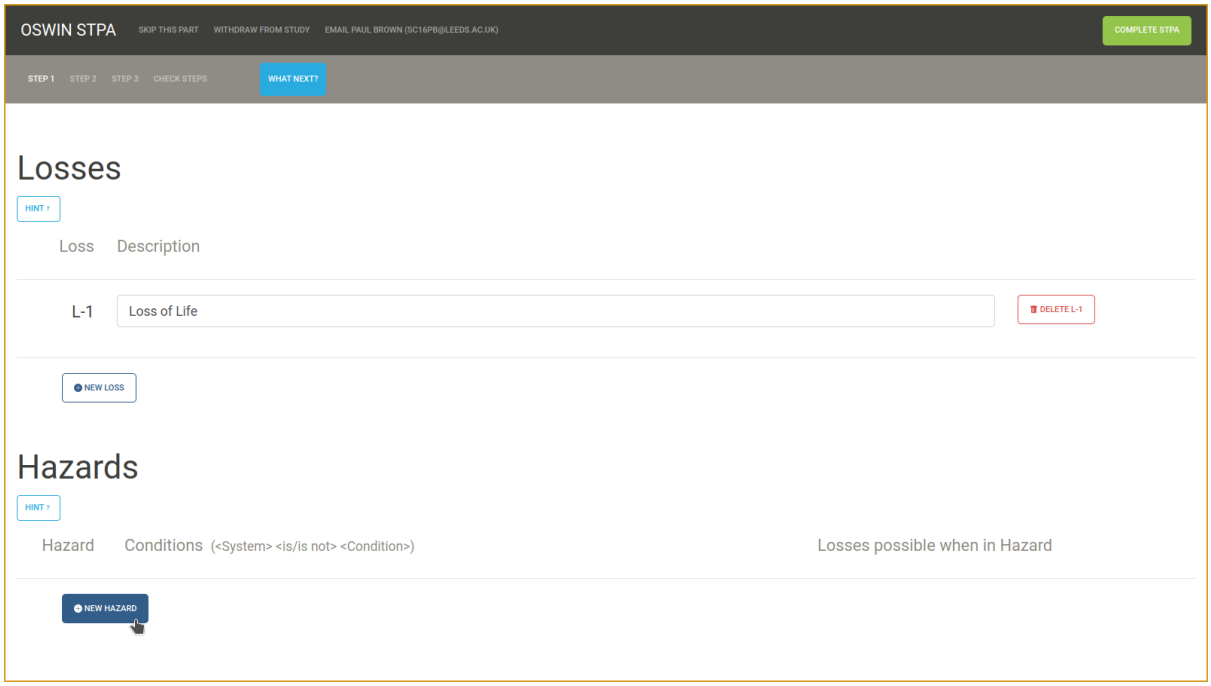

Figure 5.7: Step 1 with a loss defined.

To add a hazard the button highlighted in **Figure 5.7** is clicked. Hazards are more complex

terms, with conditions that hold and links back to losses, therefore the interface is more complex by necessity, as shown in **Figure 5.8**. In this screenshot a condition has been added which here states "H-1: Power is On", but in the ontology this has added `H-1` as a subclass of `Hazard`, created a "Power" entity, created an "On" quality, and asserted that there is a fluent holding in `H-1` with this entity and quality. Thus a total of 6 triples have been added to the ontology by this simplified representation, easing the burden on the analyst and restricting scope for error.

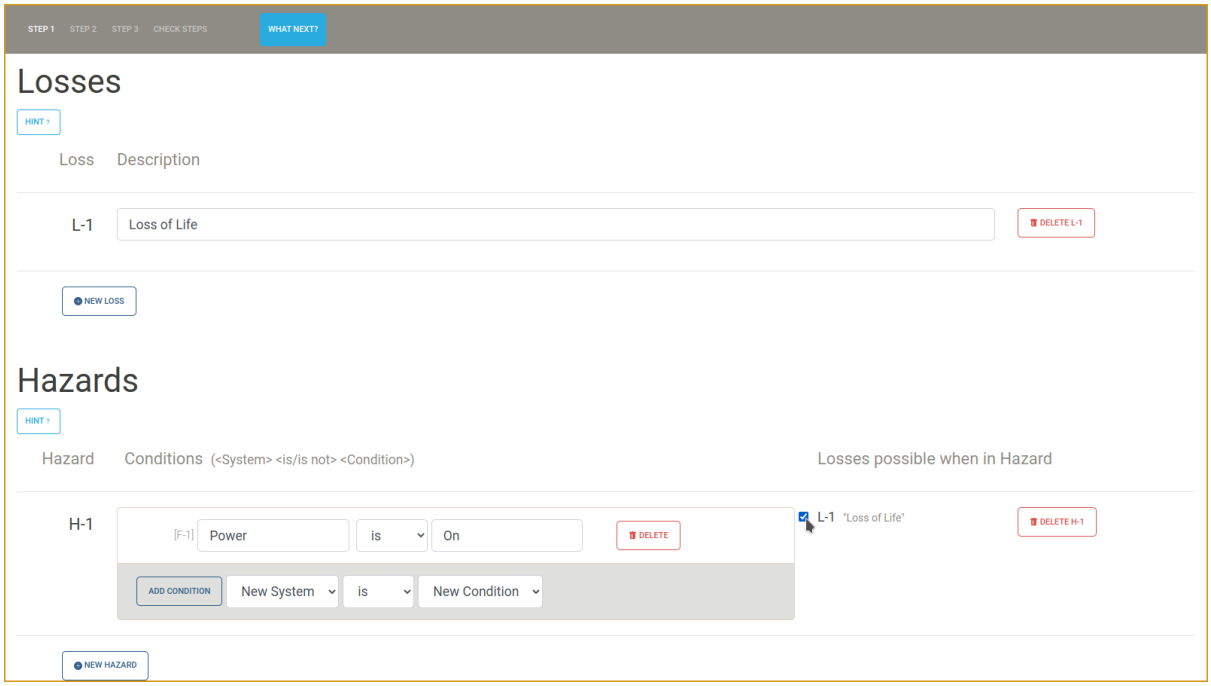

Figure 5.8: Defining a Hazard in Step 1 with condition(s) and link back to the L-1 loss.

5.5.2 Intervention Interaction Diversion

Once Losses and Hazards are defined the user may move on to step 2. However, when the user moves on to step 2 without defining any losses or hazards, Bede records the action of moving to step 2, Hilda answers OSWIN's intervention-situation-query so that OSWIN then triggers the instantiated intervention, as shown in **Figure 5.9**. This is a level 1 intervention, which is a prompt. The interventions serve as nudges, as such they're designed to be noticeable but inobtrusive. Therefore the GUI-frameworks message flashing feature is used for them, which does not block the user from continuing with other actions.

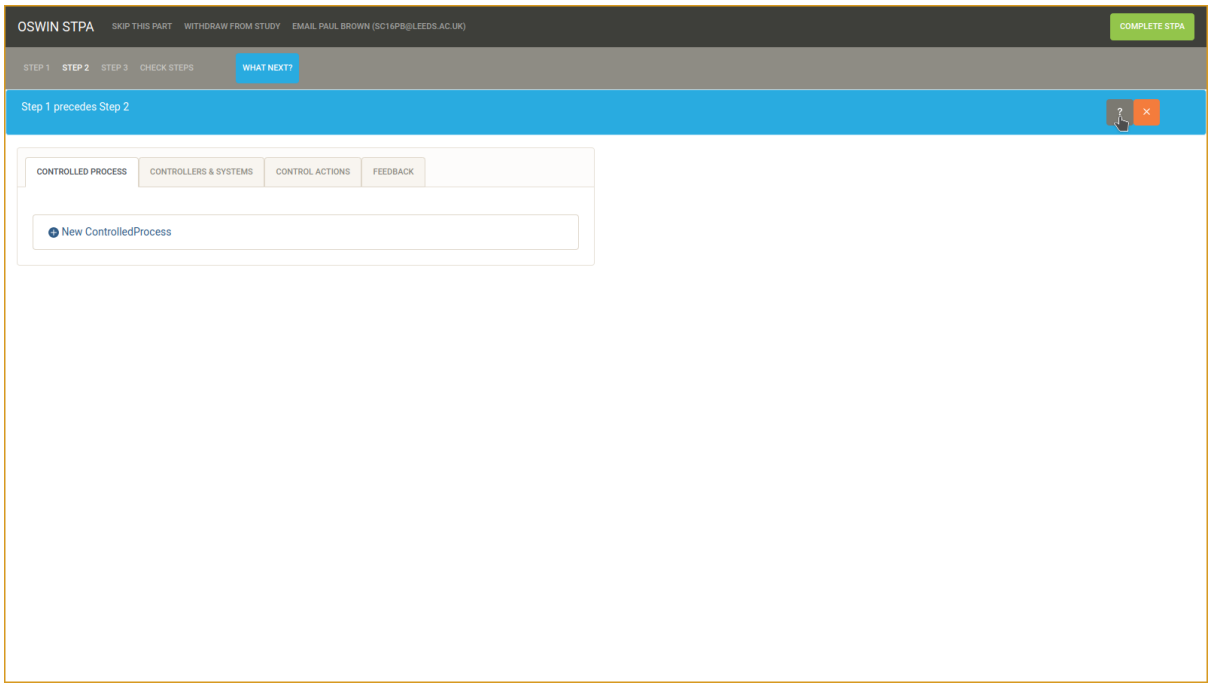

Figure 5.9: An intervention has been triggered due to no losses or hazards being defined in step 1. This is a level 1 prompt intervention.

At this point the user may navigate themselves back to step 1 to tackle the issue themselves, they may dismiss the intervention by clicking the "X" button next to the mouse cursor in **Figure 5.9**, or they can click the "?" button under the mouse cursor in **Figure 5.9** to increase the level of intervention. Again Bede records the action of the user requesting help and OSWIN queries Hilda to see if any interventions are required. OSWIN finds the current intervention in progress and the result of the query regarding which level to present the intervention at will be level 2: a leading question, shown in **Figure 5.10**. Hilda has unified the variables in the query, therefore OSWIN has all the required information to formulate the leading question from the interventions template.

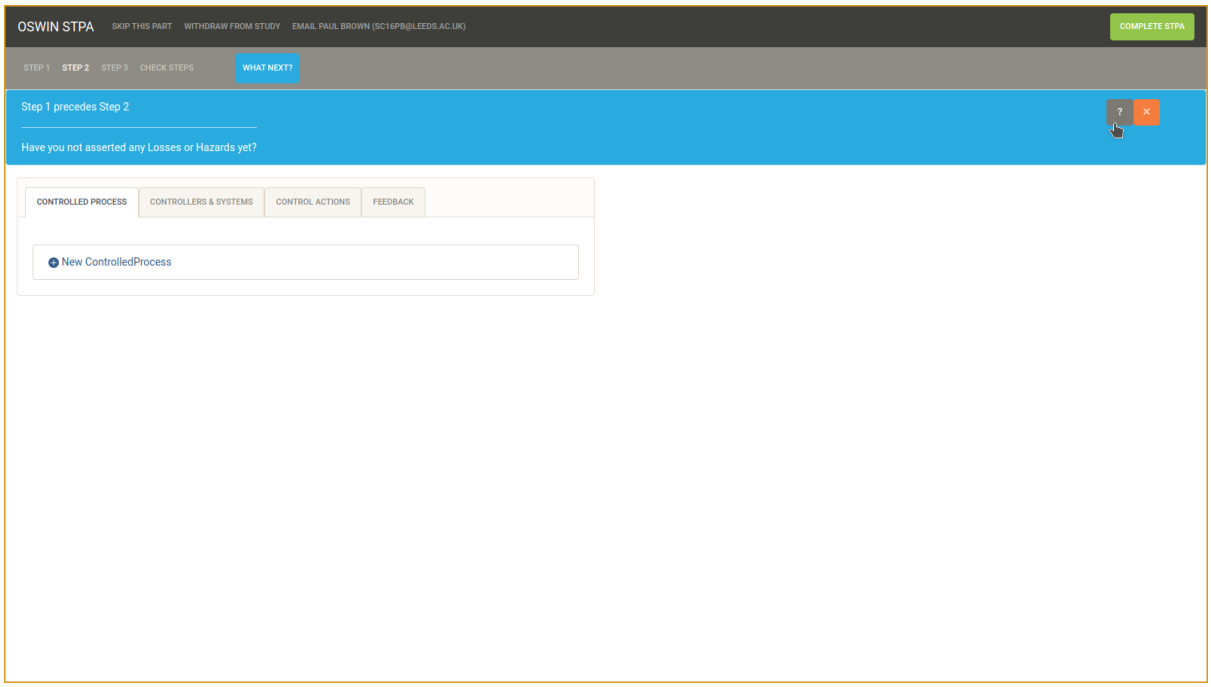

Figure 5.10: The level of intervention has been increased to a level 2 leading question at the request of the user.

In **Figure 5.10** the user has the same options as in **Figure 5.9**: begin working on their solution, dismiss, or request further help. **Figure 5.11** shows what happens when they request help: the level of intervention is increased to level 3: a tactical instruction.

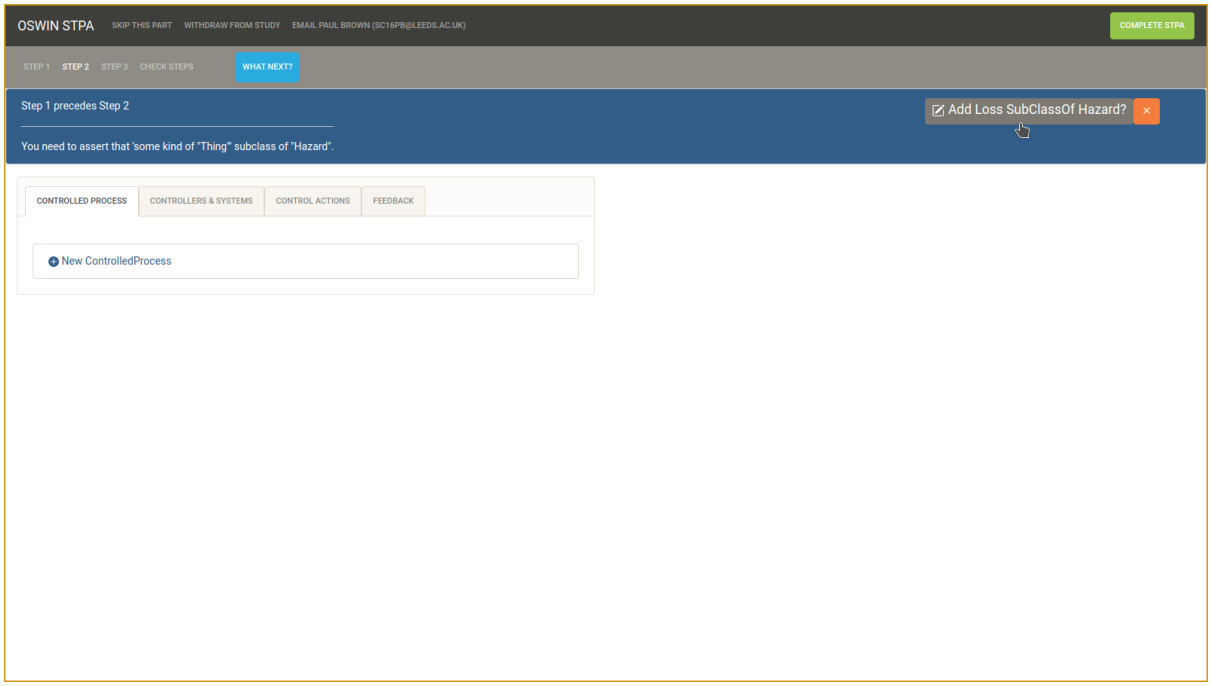

Figure 5.11: The level of intervention has been increased to a level 3 instruction at the request of the user.

At level 3 the user's option to request more help is replaced by a button, under the mouse cursor in **Figure 5.11**, that will execute some instructions in the interface to "physically" intervene. In this case the user needs to add some loss or hazard, but the software cannot know what these losses or hazards are. Therefore the most "physical" aid it can provide is to take the user to the correct place and repeat the instruction. The result of this is shown in **Figure 5.12**.

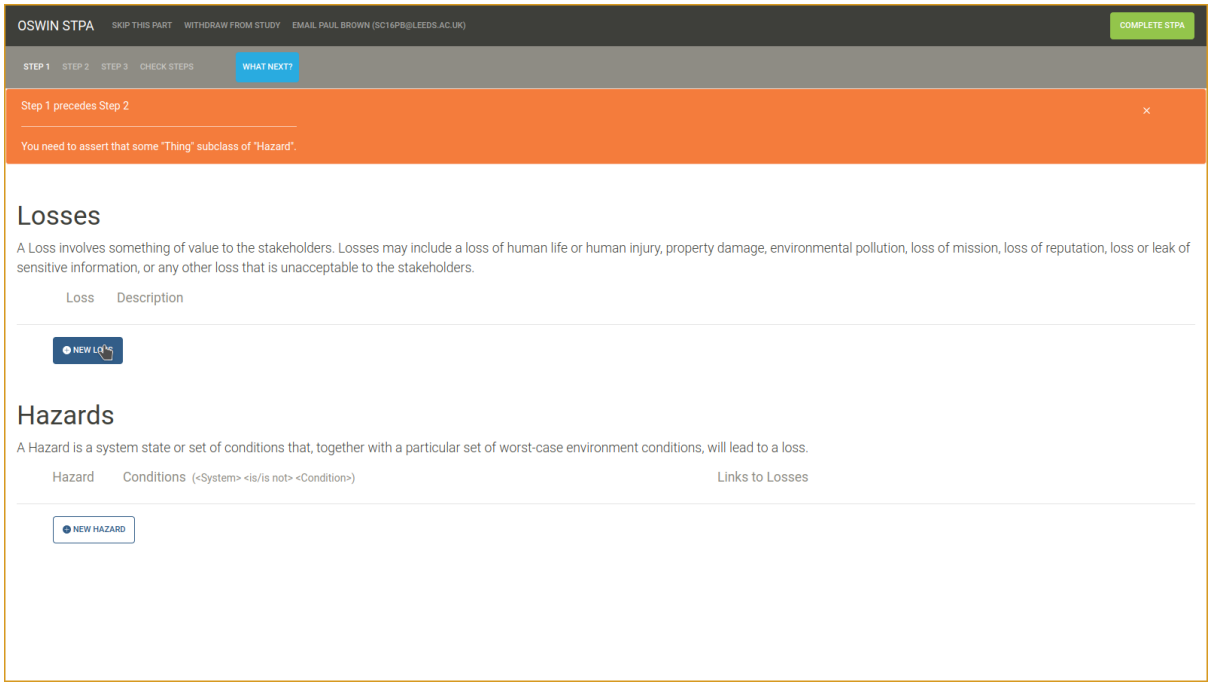

Figure 5.12: The level of intervention has been increased to a level 4 instruction at the request of the user. The software has navigated to the correct place to assert a loss for the user.

After a level 4 intervention no further help is available.

5.5.3 Step 2 Interface

With some losses and hazards defined, the user moves onto step 2 where they define the terms to automatically generate the control hierarchy diagram. Any entities they defined during step 1 while defining fluents are already in the ontology and so will already be displayed in the generated diagram. In **Figure 5.13** a "Power" and "Door" were already defined during step 1.

Step 2 requires defining multiple kinds of things, and so the analyst is given a tabbed editing panel that groups these into "Controlled Process", "Controllers and Systems" (entities), "Control Actions", and "Feedback". They do not need to be defined in order, the grouping merely serves as an organisational structure.

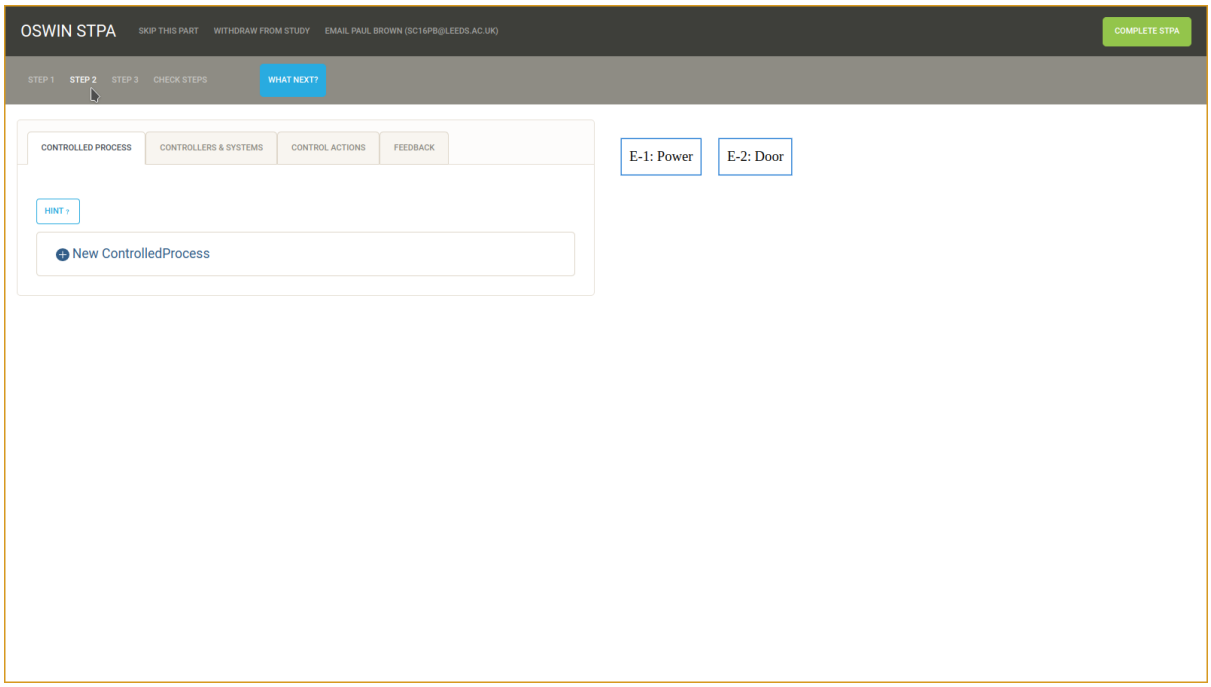

Figure 5.13: Step 2 Interface with a predefined "Power" and "Door".

In **Figure 5.14** a `CP-1` Controlled Process has been added with the label “Maintenance”. To add additional relations where `CP-1` is the subject a dropdown menu is provided.

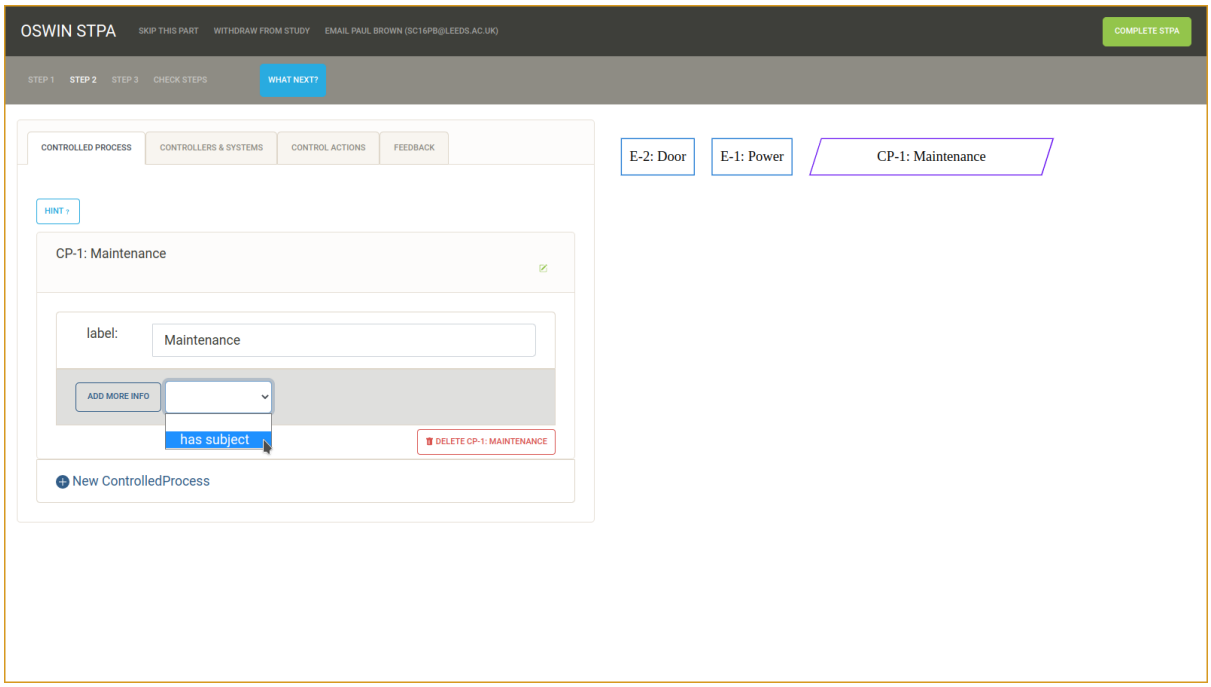

Figure 5.14: Adding a Controlled Process and information.

Using the dropdown is how “Door” and “Power” are added as subjects of `CP-1`, note they are then hidden from the diagram as they are added.

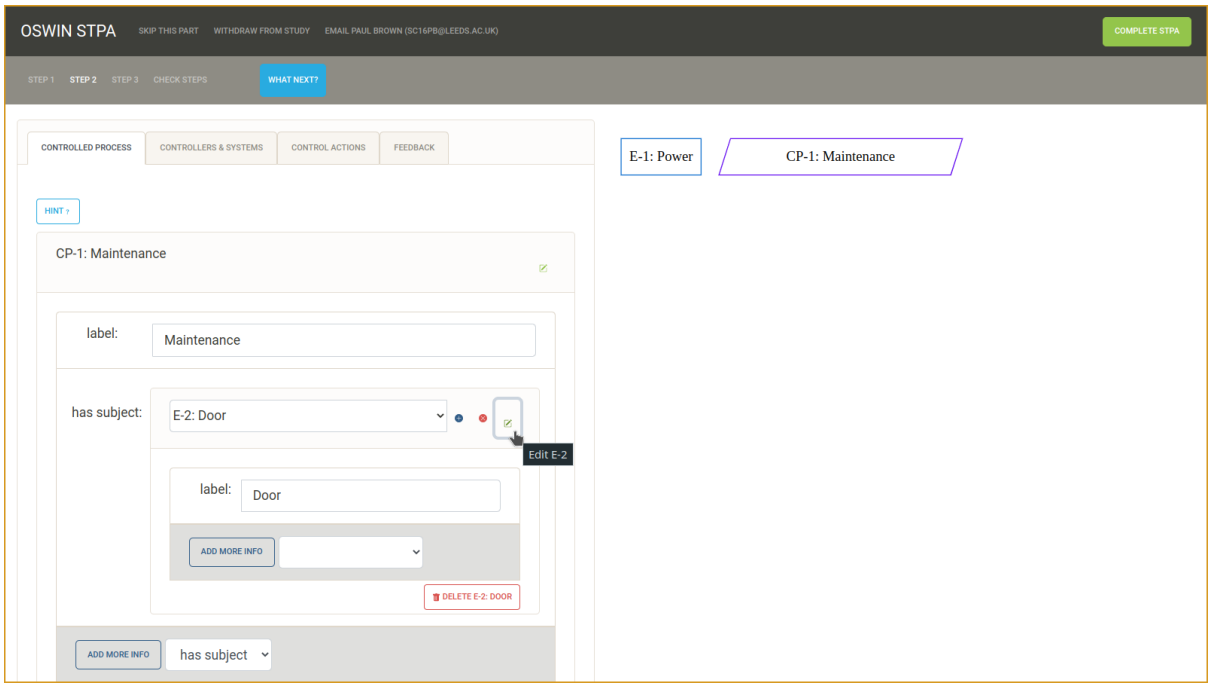

Figure 5.15: Adding related terms to a subject.

In the same way as a Controlled Process was defined in **Figure 5.14**, new systems can be defined as shown in **Figure 5.16**. This is also where the `E-1` "Power" and `E-2` "Door" entities whose existence was derived from the hazard definitions are provided for editing.

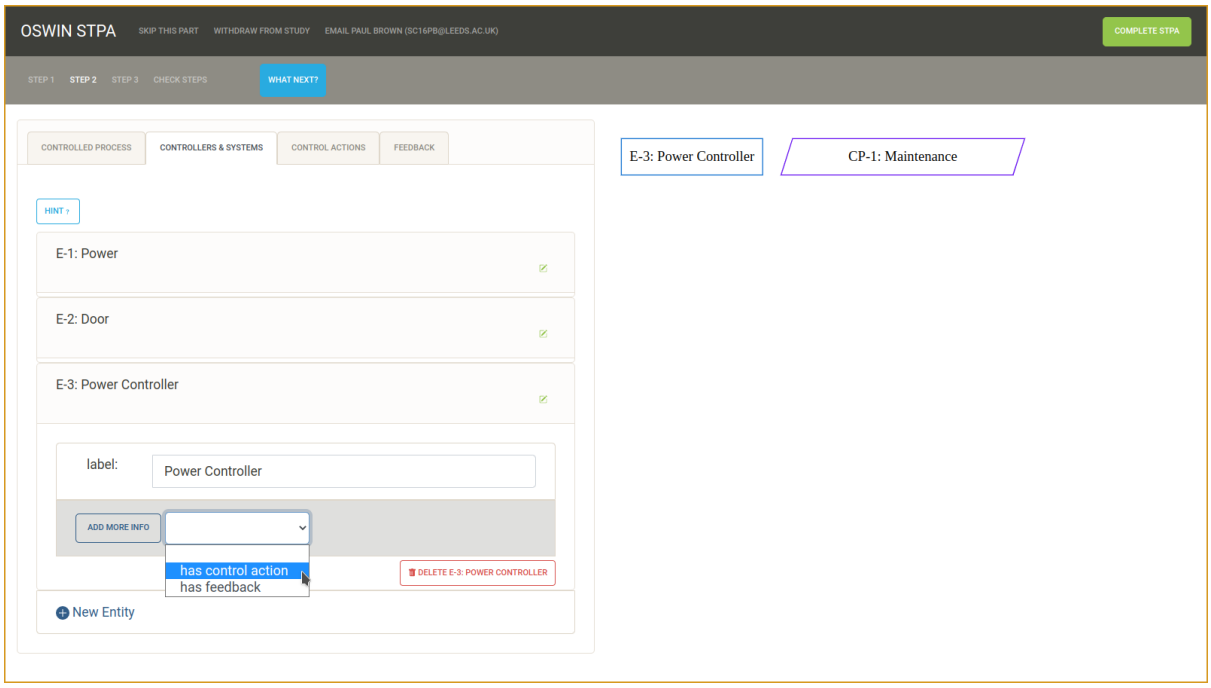

Figure 5.16: Editing entities, including those derived from hazard-fluent definitions.

The relations that can be added for the entities are not yet defined, so these are added under

their respective tabs. In **Figure 5.17** a control action is being defined with what fluents it requests hold.

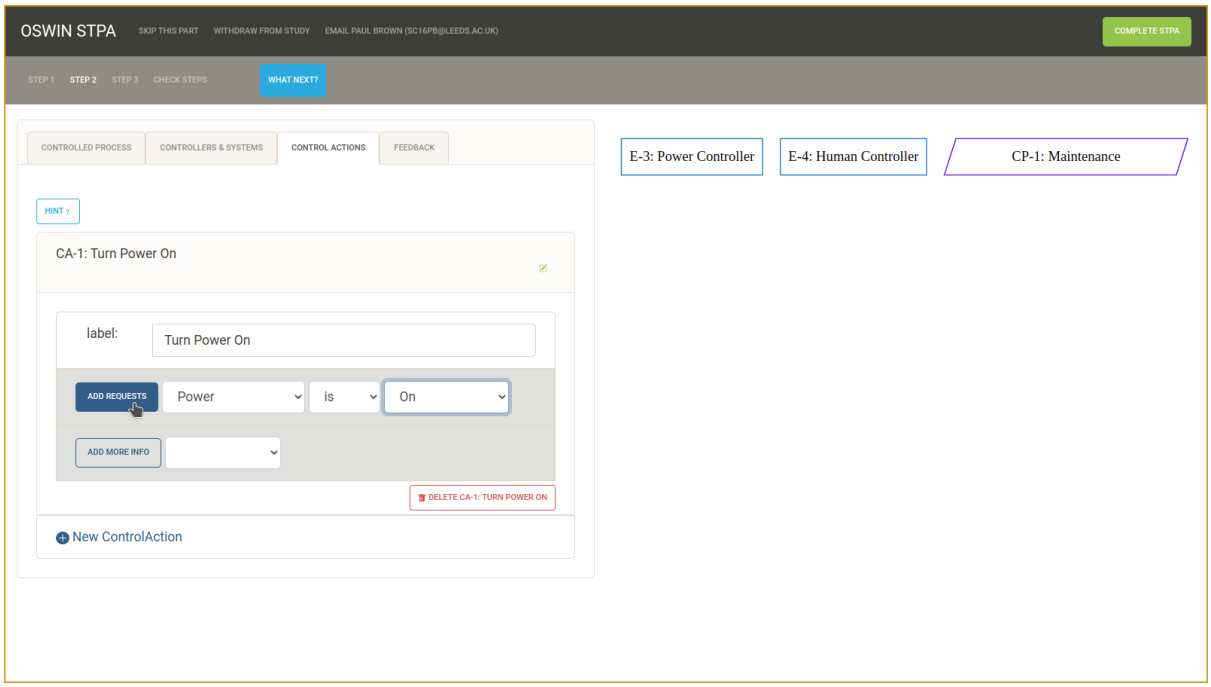

Figure 5.17: Defining a Control Action

In **Figure 5.18** the situation in which the control action is possible in is being edited. There is no distinct interface for situation terms, they are only edited as nested terms in this manner.

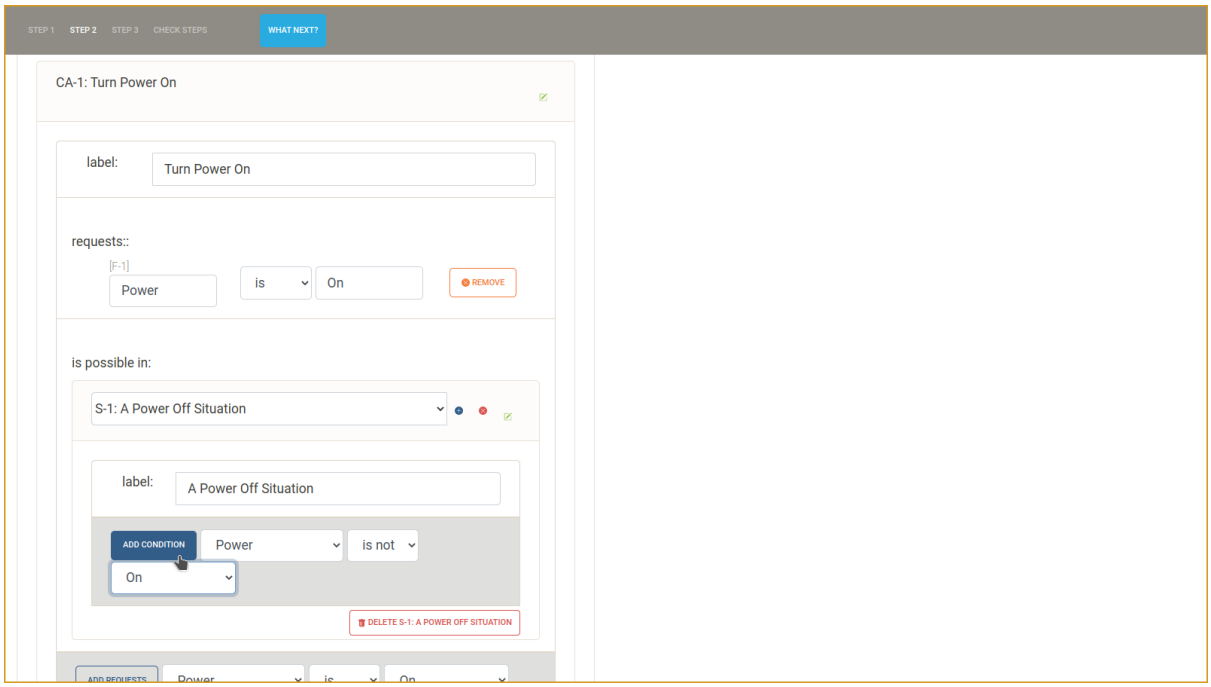

Figure 5.18: Nested editing to define the is possible in situation.

The Control Hierarchy Diagram is automatically generated from the facts asserted by the analyst, in this user-interface they have no control over how it is drawn besides by changing their asserted facts. Originally the diagram structure was determined by a query to Hilda, however by using Datalog this query was unnecessarily reproduced on the client-side to save server resources. By asserting that the "Human Controller" has the control action "Open Door", the labelled arrow representing this fact is added to the diagram in **Figure 5.19**.

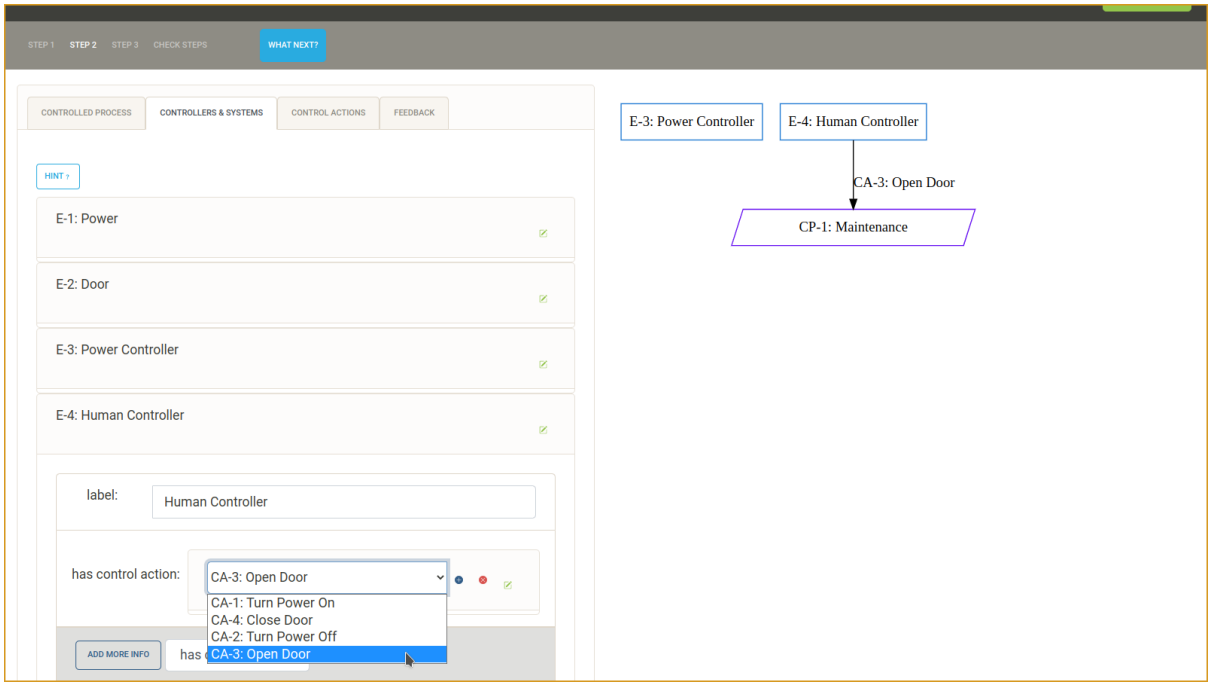

Figure 5.19: Asserting who has what control action to add arrows to the diagram.

Using different graphical user interface technologies would certainly make it possible to assert facts from the user interactively drawing the diagram, although some editing interface would still be required to add additional information such as the fluents requested by a control action. This feature was not provided only due to the drawing libraries in use not supporting interactive drawing.

Feedback is defined in the same manner as all the other terms in step 2, as shown in **Figure 5.20**.

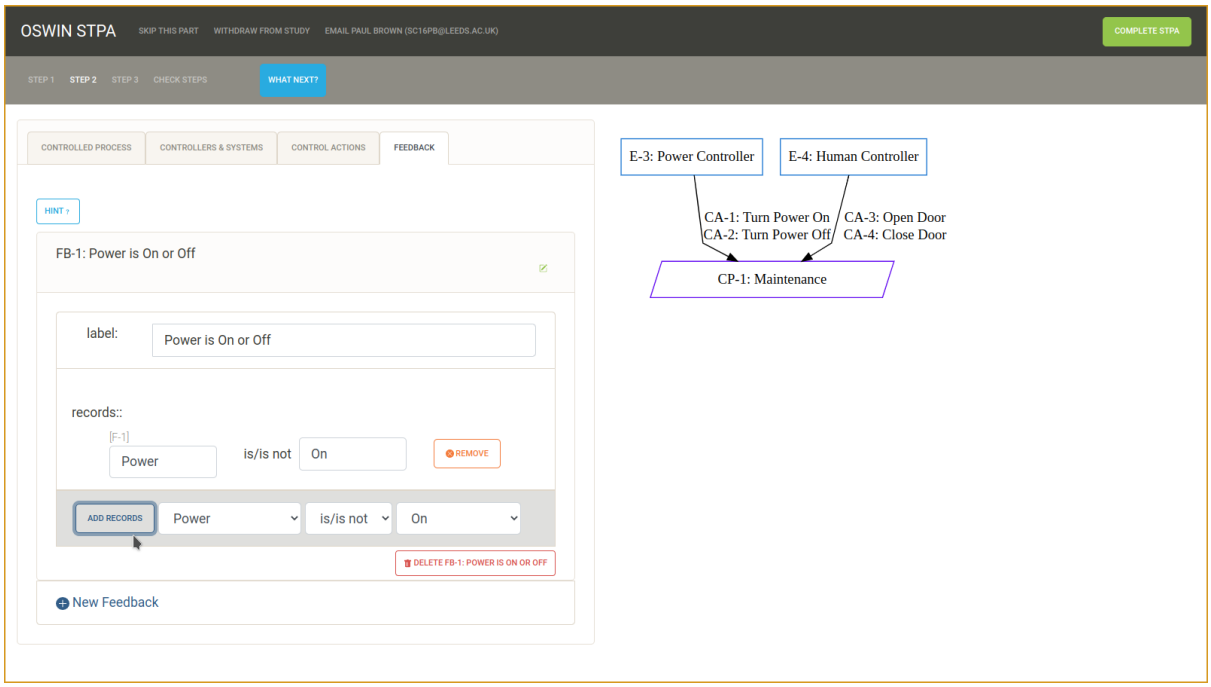

Figure 5.20: Defining Feedback

Feedback is related to the controllers in the same manner as the control actions seen in **Figure 5.19**, and so the Control Hierarchy Diagram shown in **Figure 5.21** is completed and it's time to move onto step 3.

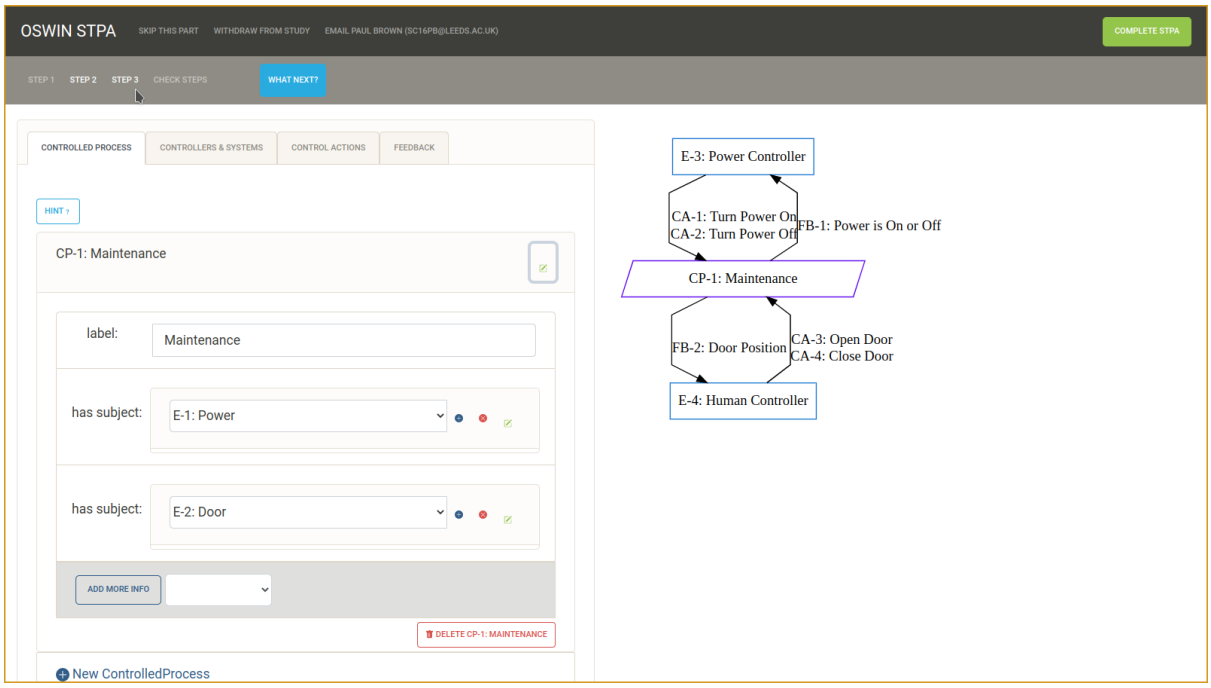

Figure 5.21: The completed Control Hierarchy Diagram, generated from the information defined by the analyst.

5.5.4 Step 3 Interface

In step 3 the analyst is required to denote which control actions are potentially hazardous when provided or not provided. This is a reduced problem from the standard STPA where the analyst also has to consider if control actions of differing durations and timing. The simplification is used as a teaching step for the STPA beginners who are the target user-group of the user-study. However, when including these other kinds of potentially hazardous control actions the update to the user-interface is not as simple as adding additional columns for these kinds because under the ontological definition a control action done for a different duration (i.e. different fluents requested to hold) or with different timing (i.e. possible in a different situation) are different control actions and so they need to be defined as such.

In **Figure 5.22** the Control Actions defined in step 2 are displayed. The analyst has the option to check the box denoting that if that control action were/were not provided then it would be potentially hazardous. Each checked box triggers an update of the ontology, which is recorded by Bede.

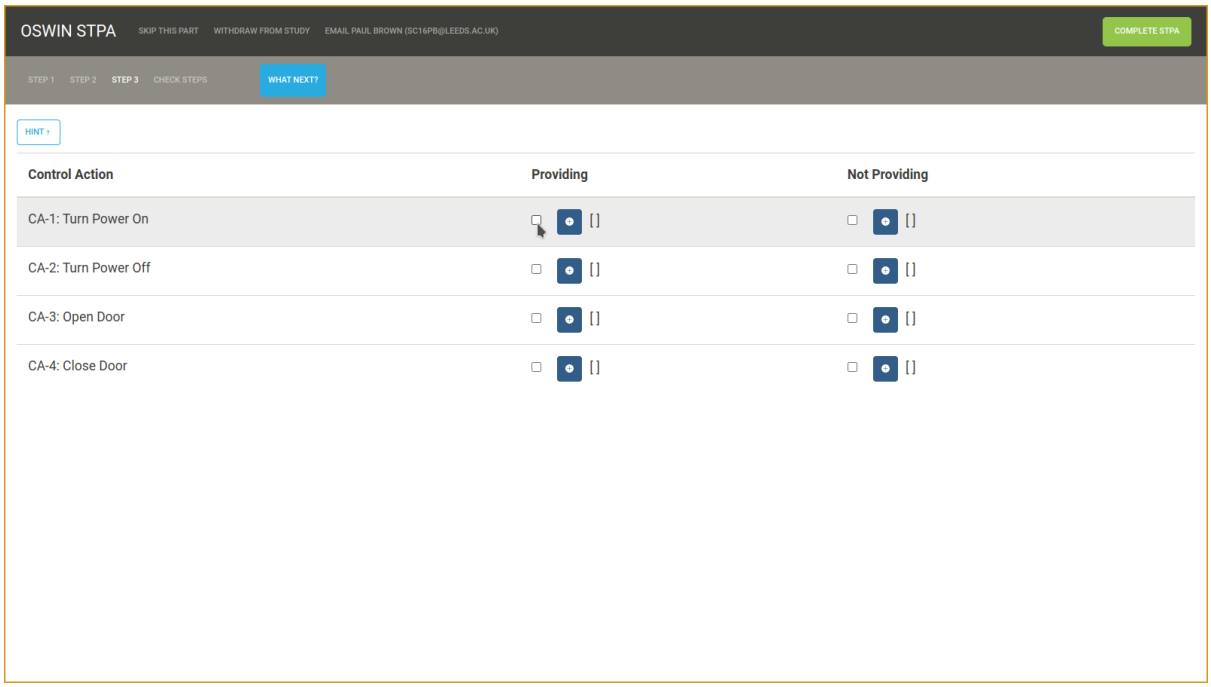

Figure 5.22: Step 3: Denoting if a control action is potentially hazardous if provided or not provided.

A potentially hazardous control action should also be defined with the situation in which it is possible, and it should be linked to the hazard that it may lead to. The situation in which it is possible in is already provided for and shown in **Figure 5.18**. To allow an analyst to link the

hazard the "+" button and pop-up are provided shown in **Figure 5.23**.

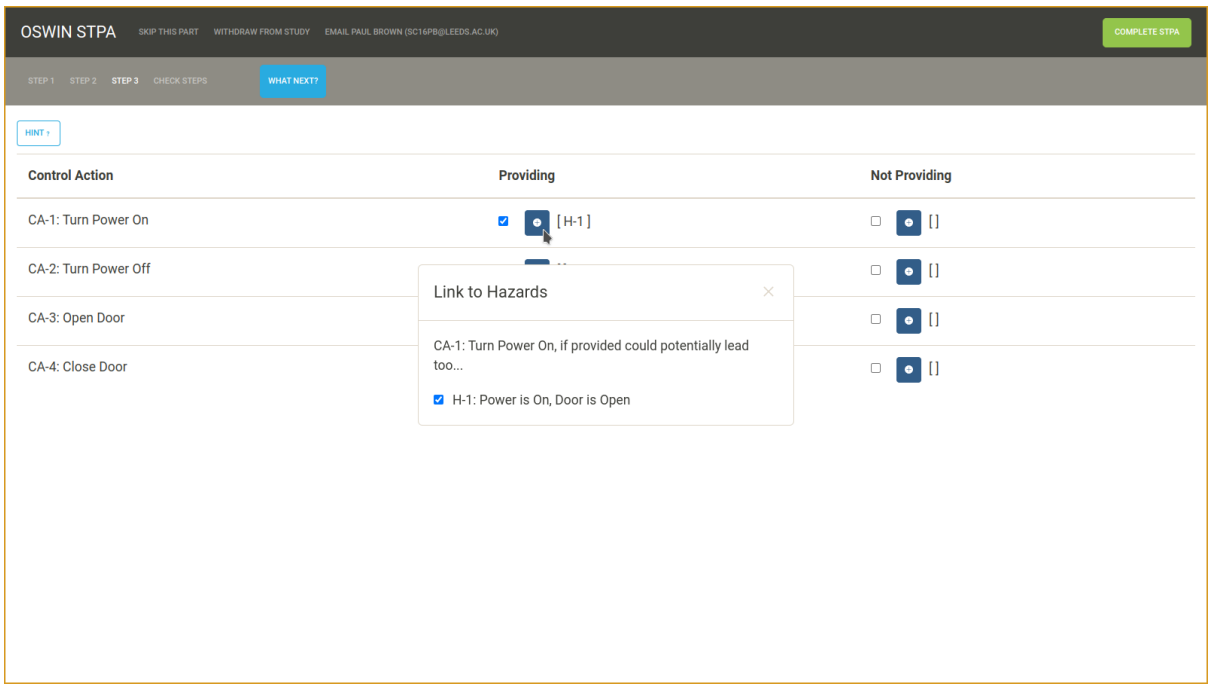

Figure 5.23: Linking to Hazards

This completes the interface for step 3, which completes the requirements for the STPA analysis by the target user-group for the user-study.

5.5.5 What's Next? A Proactive Interface for Seeking Help

The missing-type of interventions check to see if some definitions are included in the ontology. They are only checked after the relevent step, otherwise on starting the analysis a user will be presented with a barrage of advice that may not be welcome or sensible at that time. To aid a user that may not be sure of what action they should take next, the missing-interventions for the current step are made available via the "What's Next?" button. Clicking on that button will provide such an intervention if found, as shown in **Figure 5.24**. It works like a "What If?" kind of question that Situation Calculus was designed to answer (See **Section 2.2.2**), OSWIN asks "What if we were in the next step? Then what interventions would there be?", which it does by temporarily (and in memory only) moving onto the next step in the current situation before querying Hilda for interventions.

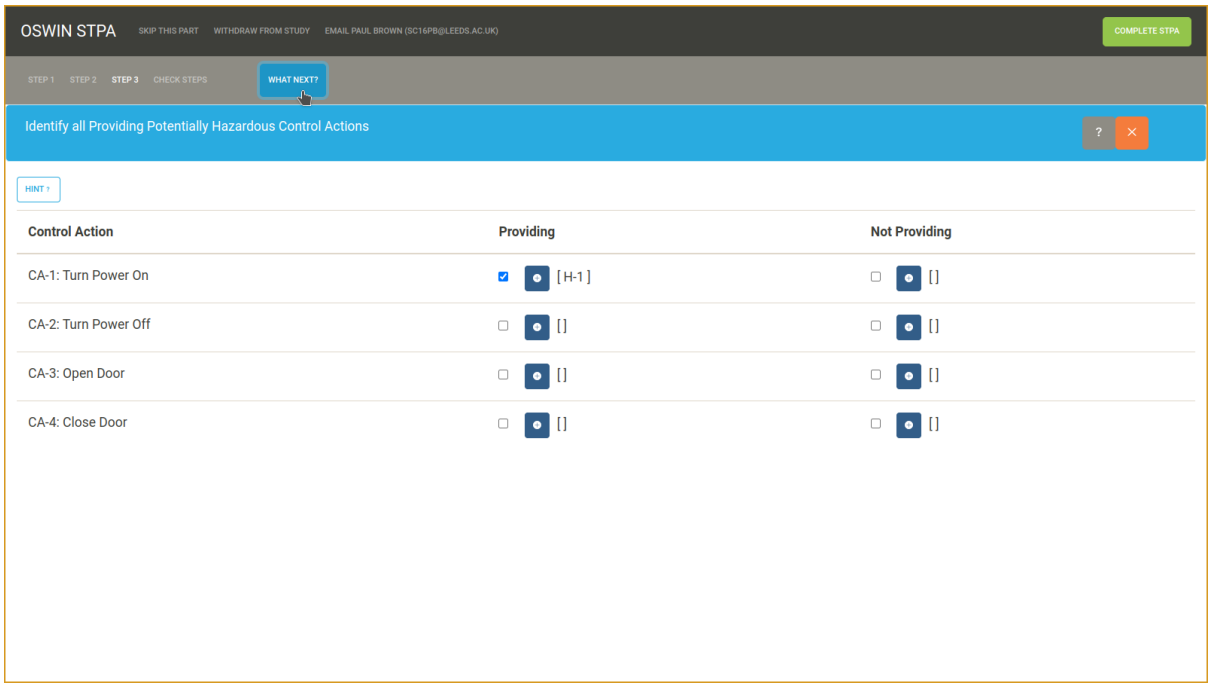

Figure 5.24: Clicking on "What's Next" shows a missing-type of intervention for the current step if one such intervention is possible.

These interventions behave in the same manner as regular interventions, **Figure 5.25** shows the level of the intervention being incremented to level 2, just like **Figure 5.10** shows for a regular intervention.

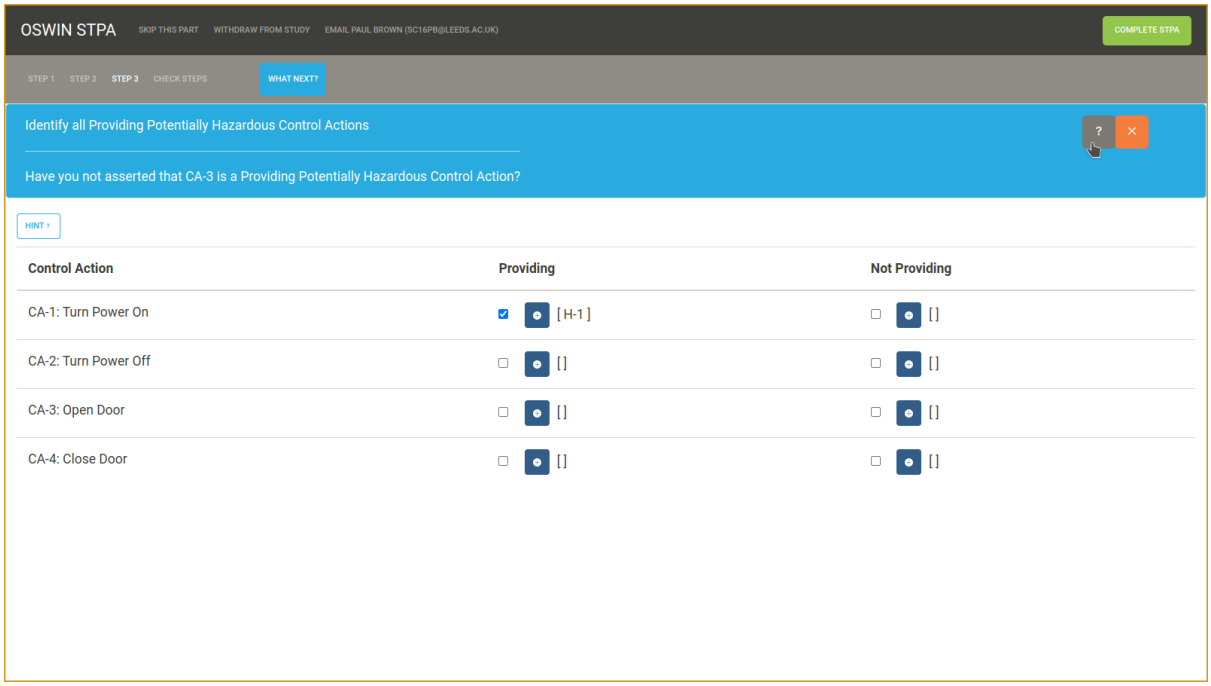

Figure 5.25: Incrementing a "What's Next?" intervention in the same way as a regular intervention.

It should be noted that the intervention shown in **Figure 5.25** demonstrates the capability of the reasoning in the software to automatically determine whether providing or not providing an *as defined* control action is potentially hazardous. However, to do so in this application would be contrary to the intended goal, which is to aid in learning STPA rather than completing an STPA.

5.6 Intervention Walk-Through Examples

In this section examples taken from the user evaluation logs will be described to show how the intervention system works. The two examples include both missing and mistake interventions, successful and required interventions.

5.6.1 A Simple Mistake Quickly Resolved

This is from the log file of user 36, beginning with the 147^{th} action about 10 minutes into their analysis.

```
focus_concept('Feedback',36,
        datetime(2022,3,8,16,44,9)).
assert(spo('FB-1',label,"Module evaluation feedback"),36,
        datetime(2022,3,8,16,44,19)).
assert(spo('FB-1',recordsFluent,'F-3'),36,
        datetime(2022,3,8,16,44,30)).
intervene(
  'Avoid using ambiguous and vague labels in the control structure:
  "commands", "feedback", "status", "computer"',
  logged_assertion('FB-1',subClassOf,'Feedback') and
  logged_assertion('FB-1',label,"Module evaluation feedback") and
  fact("Module evaluation feedback",containsWord,feedback) and
  fact(feedback,instanceOf,'AmbiguousOrRecursiveWord'),
  1,datetime(2022,3,8,16,44,30)).
```

The user has navigated to the Feedback tab and asserted some "Module evaluation feedback" exists and records the fluent 'F-3'. The STPA Handbook (N. Leveson and Thomas 2018, p.31) advises against using the word "feedback" when defining feedback as being vague and ambiguous,

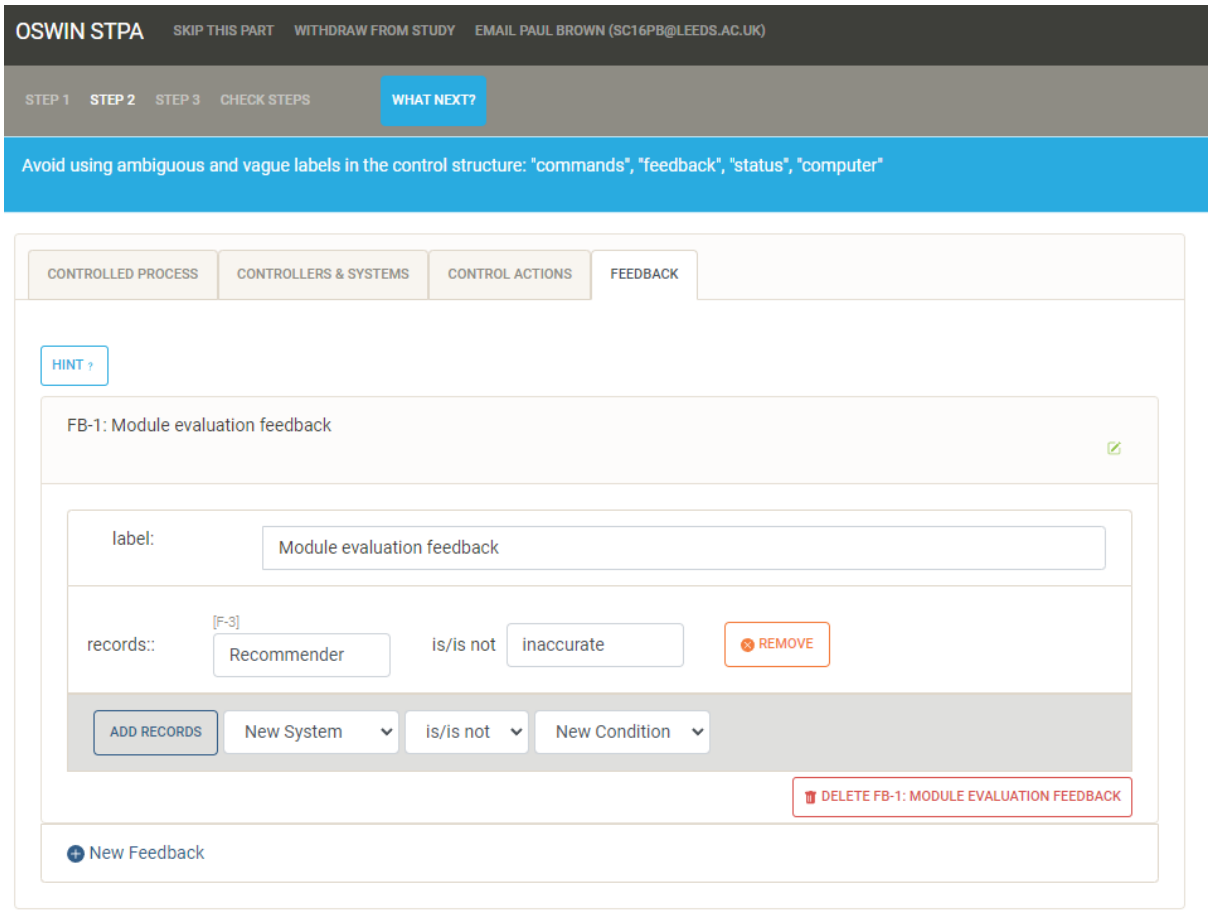

Figure 5.26: Intervention regarding advice about label triggered for user 36

therefore this is included as an intervention. The second term with the `intervene/4` term in the log is the intervention query that succeeded in order to fire this intervention, which has been given as a level 1 intervention as it has not fired before. This can be seen as presented to the user in the screenshot in **Figure 5.26**.

The user then takes 15 seconds to read the intervention, consider it, and respond:

```
retract(spo('FB-1',label,"Module evaluation feedback"),36,
    datetime(2022,3,8,16,44,45)).
assert(spo('FB-1',label,"Module evaluation survey results"),36,
    datetime(2022,3,8,16,44,45)).
```

This label change removes the word "feedback" and so the intervention is resolved. It's also improved the label of 'FB-1' in the manner intended: before it wasn't clear how the module would be evaluated, the new label is less vague and ambiguous and the system designers can consider the implications of an evaluation survey rather than some unknown method of evaluation.

Immediately after resolving the intervention, which vanishes from the UI, the user click's the "What Next?" button to request an intervention. As the one they were working on was resolved the AI system looks for a "missing" intervention and finds one to present. The user doesn't engage with this intervention, suggesting their request was for confirmation of the validity of their correction rather than as a request for help.

```
request_intervention(36,datetime(2022,3,8,16,45,7)).
intervene('Step 2 precedes Step 3',
  not (logged_assertion(_23438,hasCapability,_23442) and
  fact(_23442,requestsEffect,_23498) and
  logged_assertion(_23530,subClassOf,'Feedback')),
  1,datetime(2022,3,8,16,45,7)).
focus_concept('ControlAction',36,datetime(2022,3,8,16,45,10)).
focus_concept('Feedback',36,datetime(2022,3,8,16,45,11)).
focus_concept('ControlAction',36,datetime(2022,3,8,16,45,13)).
focus_concept('Feedback',36,datetime(2022,3,8,16,45,14)).
focus_step(step3,36,datetime(2022,3,8,16,45,16)).
```

The log shows they survey step 2 terms before moving on to step 3. If this is discovered to be a repeated pattern, that may suggest users are expecting confirmation when resolving an intervention that's not being provided.

5.6.2 Missing Relation and Missing Intervention

This example is from user 43's log and the interaction begins about 27 minutes and 241 actions into their analysis. By this point they've already defined much of their control hierarchy terms for step 2 and they're making use of the AI to guide their workflow. So they request an intervention, and the AI identifies that their controlled process isn't controlled by any controller. In this case they're simply missing some `hasSubject` relationships.

```
request_intervention(43,datetime(2022,3,9,13,17,19)).
intervene('Check that every controlled physical process is
  controlled by one or more controllers',
  logged_assertion('CP-1',subClassOf,'ControlledProcess') and
  not (logged_assertion('CP-1',hasSubject,_11750) and
```

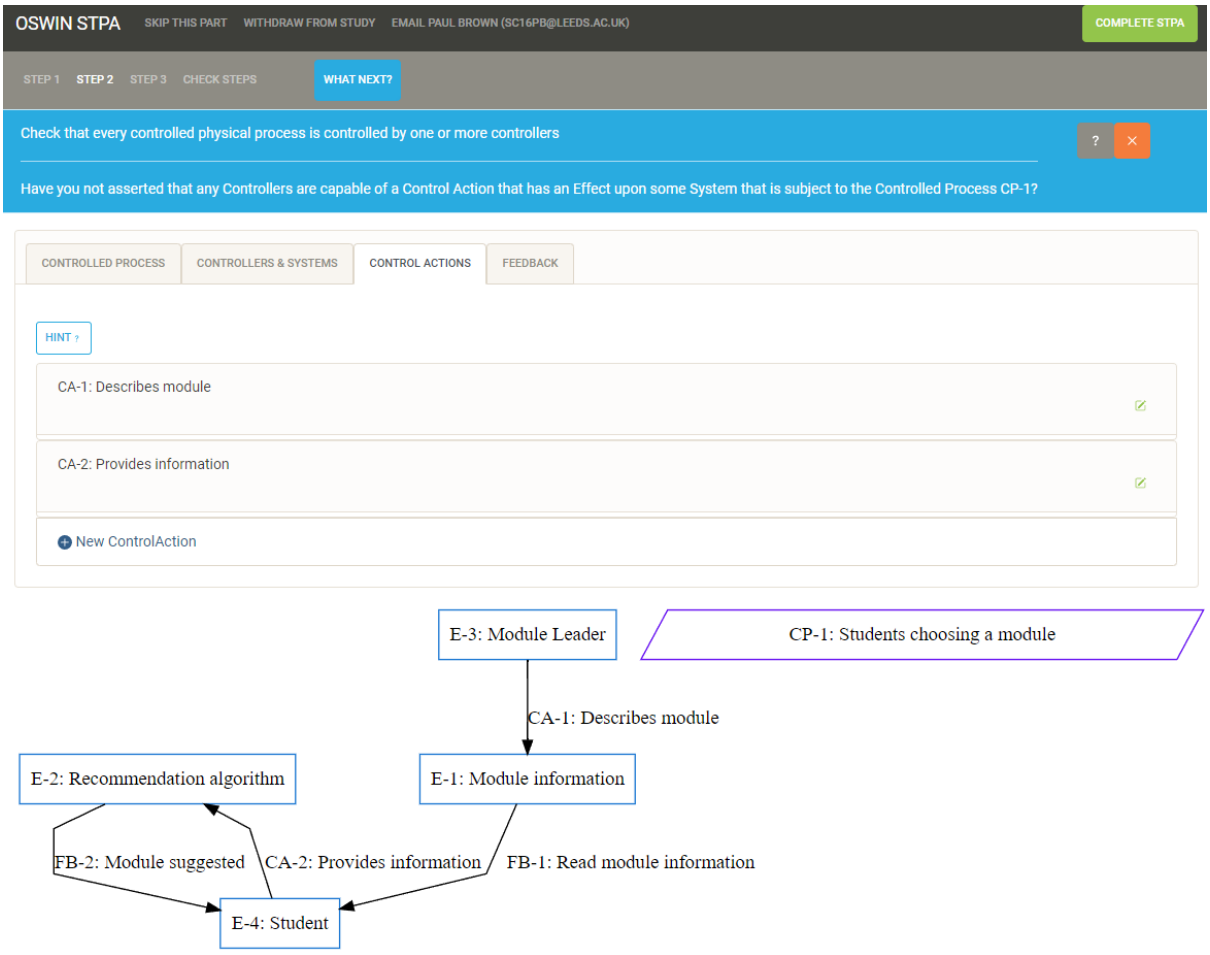

Figure 5.27: Intervention regarding an uncontrolled "Control Process" (CP-1) for user 43.

```
    logged_assertion(_11802,hasBearer,_11750) and
    fact(_11858,requestsEffect,_11802) and
    logged_assertion(_11894,hasCapability,_11858)),
  1,datetime(2022,3,9,13,17,19)).
request_help('Check that every controlled physical process is
  controlled by one or more controllers',
  43,datetime(2022,3,9,13,17,25)).
focus_concept('ControlledProcess',43,datetime(2022,3,9,13,17,39)).
glossary_lookup('ControlledProcess',43,datetime(2022,3,9,13,17,41)).
focus_concept('ControlAction',43,datetime(2022,3,9,13,17,57)).
focus_concept('ControlledProcess',43,datetime(2022,3,9,13,18,5)).
assert(spo('CP-1',hasSubject,'E-4'),43,datetime(2022,3,9,13,18,14)).
retract(spo('CP-1',hasSubject,'E-4'),43,datetime(2022,3,9,13,18,53)).
assert(spo('CP-1',hasSubject,'E-3'),43,datetime(2022,3,9,13,18,53)).
retract(spo('CP-1',hasSubject,'E-3'),43,datetime(2022,3,9,13,19,4)).
assert(spo('CP-1',hasSubject,'E-2'),43,datetime(2022,3,9,13,19,4)).
request_intervention(43,datetime(2022,3,9,13,19,28)).
focus_step(step3,43,datetime(2022,3,9,13,19,30)).
```

In response to the intervention they request help, which increases the intervention level to 2 (this kind of intervention hasn't occurred previously in their log). They're then presented with the leading question: "Have you not asserted that any Controllers are capable of a Control Action that has an Effect upon some System that is subject to the Controlled Process CP-1?" as shown in **Figure 5.27**. This long question contains a lot of details to unpack, rather than request more help they instead identified, in 14 seconds, that they needed to do something about "Controlled Process" and navigated to it.

They immediately (2 seconds) checked the glossary for "Controlled Process" as they were trying to figure out what to do. After 16 seconds of thought they referred to their Control Actions, which is significant because they help inform the "hasSubject" relationship. Following this they asserted and retracted several "hasSubject" relationships as they were thinking through the possible objects before settling on their final answer.

Once they've settled on `CA-1 hasSubject E-2`, they repeat the same checking behaviour of

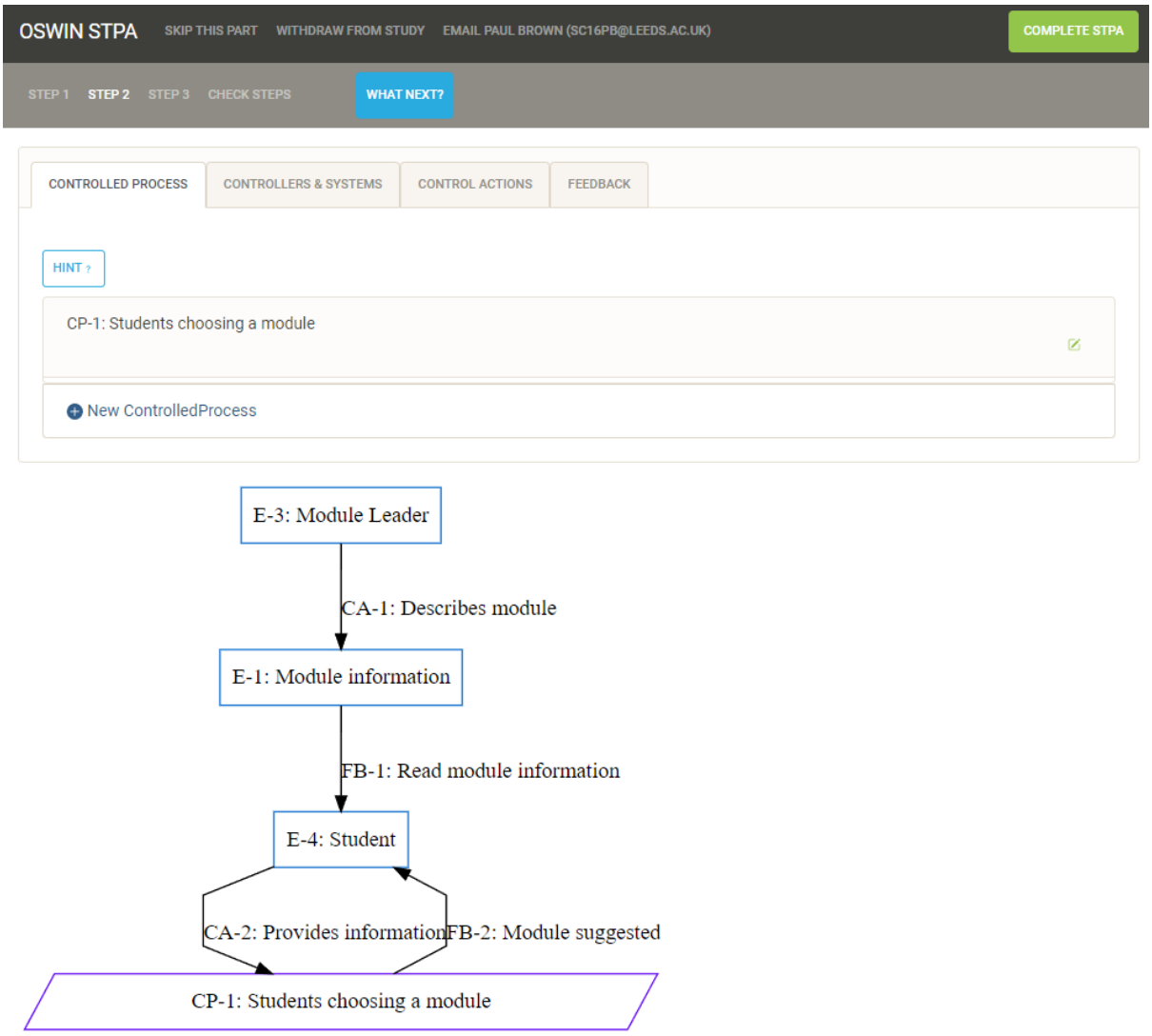

Figure 5.28: Intervention for user 43 resolved, but still incorrect as E-1 is also subject to CP-1.

hitting the "What Next?" button to request an intervention. None is found and it continues. The intervention that was resolved is to ensure that every controlled process in the model is controlled by some controller. However, there is another entity in the model which should be declared as the subject of CA-1: E-1 is controlled by E-3 and has no control actions of its own, as shown in **Figure 5.28**. This is a different intervention that was not included in the intervention bank and which became apparent during this user study: if an entity has no capability that's a control action and it's not the subject of any control process then it's likely the user has missed one of these relationships. Adding this intervention to the bank, now it's identified through observation, would be trivial.

5.7 Software Implementation Discussion

The first aim of the software implementation was to create an executable software version of the theoretical aspects of the code. The programming languages used were chosen to ease this. Firstly the STAMP ontology was defined using a subset of the $\mathcal{SHOIN}(\mathcal{D})$ Description Logic (See **Section 3.1**), which can be defined using OWL-DL. SWI-Prolog provides libraries for reading this format, and so the assertions in the ontology are the same in DL, OWL, and as Prolog facts. What differs is the reasoning, which in Prolog is with horn-clause logic. This resulted in an incomplete definition for subsumption, as discussed in **Section 5.3.3**, although this application is sufficiently restricted as to not be effected by it.

The ontology authoring and Contingent Scaffolding frameworks were both defined in Situation Calculus (**Section 4.2** and **Section 4.3.2** respectively). Therefore with the `SitCalc` library providing Situation Calculus reasoning both frameworks could be defined in Logtalk, with minor syntactic alterations such as using multiple clauses for disjunction.

The claim of these frameworks and libraries is that they are generalized and so reuseable. However in Pre-Whitby they were not able to be easily shared and reused. Taking a set of rules and applying them to some facts is a typical task in Prolog. However, the limitations of the module system often result in code that only handles a fixed set of facts at a time, either imported into the rules module or loaded into the `user` special module. But sometimes these rules are useful to many applications, as is the case with Situation Calculus. When the rules are to be shared as third-party libraries, any dependency of rules on facts needs to be inverted to decouple the rules from a particular set of facts. This dependency inversion allows multiple set of facts to

be loaded and used concurrently (providing an alternative solution for implementing the *many-worlds* design pattern). Key to this dependency inversion is the concept of *interface* or *protocol*, supported by Logtalk but absent in Prolog module systems.

This inversion was achieved in Logtalk by taking inspiration from the Abstract Factory design pattern and considering how it could be achieved with *protocols* and *categories*. The final solution is simpler than the Abstract Factory design pattern as no dynamic creation of objects is necessary. Instead, dependency upon a *protocol* and conforming to it is all that is required. This is an elegant pattern for Logtalk that can be repeated when creating third-party libraries to reason about definitions in an application without depending upon them.

The use of *protocols* in this manner results in a *plugin architecture*. A third-party can "plug-in'' code to the `SitCalc` library, or other libraries, to work with it. This is a very versatile design pattern as it allows an application developer, or even third-parties and end-users provided with a plugin loading interface, to adapt the behaviour of the application to their needs without editing the core application code. It also leaves the application immune from changes made elsewhere via the plugin, with the provision they are not malicious, by the drawing of boundaries in the architecture (Martin 2018).

This use of *protocols* has focused on their application for dependency inversion due to the specifics of the Whitby application architecture. It should be noted their use also resulted in adherence to the Single Responsibility and Open-Closed Principles. Protocols also have significant contribution to adherence to the Liskov Substitution Principle, making it a simple matter to swap objects that adhere to the same protocol, as well as the Interface Segregation Principle by providing explicitly defined interfaces as first class entities.

The refactor from Pre-Whitby to Whitby decoupled code from Pre-Whitby that can be reused, which are published as third-party libraries to satisfy the motivation behind the refactoring. This has simplified Whitby, where there is less functionality now to maintain, and has enabled other applications and libraries to use Situation Calculus reasoning while also keeping a clean architecture. The workarounds that we attempted to compensate for the lack of required features in the Prolog module systems accumulated and increased the complexity of the application. Those workarounds are not supported by development tools (especially documenting and diagramming tools) and raised new issues, thus creating additional burden on developers while not solving the reusable goals that prompted the refactoring.

By using the language constructs provided by Logtalk to apply SOLID principles in the refactoring, the Whitby application documentation and diagrams trivially reflect the actual architecture of the application, further simplifying development and maintenance. But hand-coded workarounds that try to compensate for missing language features (in this case: the module system in the original version of the application) required additional effort to document as they are not visible to developer tools as first-class constructs. These workarounds must also be repeated in every application with the impact of their limitations carefully taken into account.

This refactoring has benefited the Whitby application, the Situation Calculus reasoning is open to extension without modification, which was used to add application specific fluents and actions as the need arose. Additionally, the separation of responsibilities has made it easier to navigate and edit the code base. But the primary benefit is to other applications that wish to make use of the extracted libraries. Whitby demonstrates how they can be reused. Bedsit[8] is one example of such reuse: it is an exploratory framework for rapidly prototyping applications using `SitCalc` and includes both TicTacToe and ToDo example applications with a variety of UIs. The author has also reused `SitCalc` and `OntAuth` to quickly prototype a proprietary ontology browser and editor.

As part of the AI4EU[9] initiative, Rubinstein Pérez (2021) was provided with Whitby to adapt to a new project in the domain of robotics planning. Due to Whitby's adherence to the Single Responsibility and Liskov Substitution Principles, which was not possible with Pre-Whitby, a third-party should need only to make changes at the periphery of the code base: telling `kb` to load a different OWL file, optionally substituting any reasoning rules specific to their domain, substituting `intervention_bank` for an object with appropriate interventions, and substituting the `Editor` GUI to be appropriate for that domain. There is still room for improvement, however. For example, they also needed to change a list in `action_bank`, which contains the classes used as tabs in the GUI.

In addition to demonstrating the generalisability and reuseability of the code, and thus the definitions from which the code was authored, there is also an important issue of timing. Contingent Scaffolding depends upon offering support as soon as the learner is in trouble, and so the time to find and return a relevant intervention must be "immediate". This is primarily achieved through

[8] `https://github.com/PaulBrownMagic/BedSit`

[9] `https://www.ai4eu.eu/` Established to build the first European Artificial Intelligence On-Demand Platform and Ecosystem with the support of the European Commission under the H2020 programme.

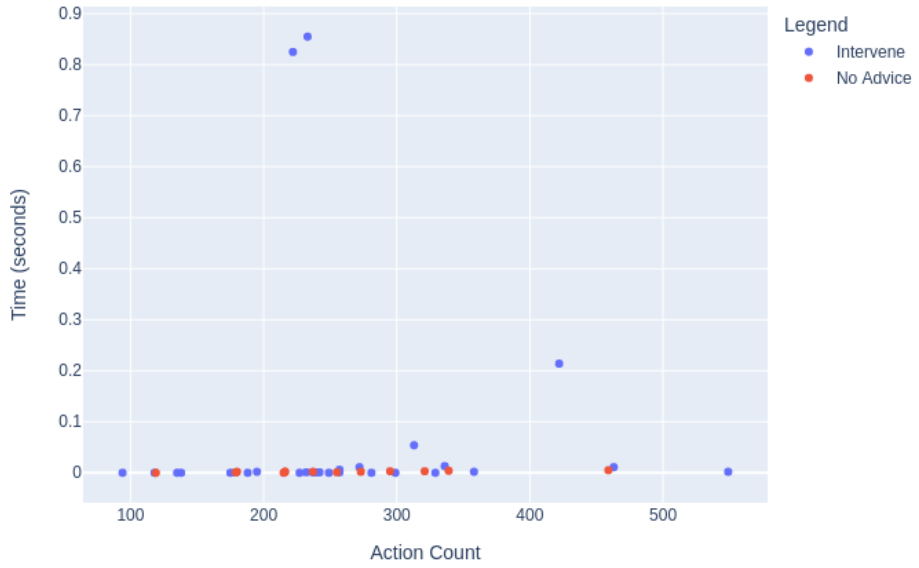

Figure 5.29: Time to find an intervention for the logs final situation after the immediately prior situation has been queried so that memoization with tabling is exploited. Testing was done on a single core of a CPU with 512KB cache size and running at 3792.914 MHz.

tabled-resolution in Prolog to memoize reasoning already done. Furthermore the number of interventions returned is restricted to one[10] and returning any on-going intervention that's still relevant before searching for another. With these measures in place the time taken to intervene has been reduced to less than $1/10^{th}$ of a second for the majority of final situations with a maximum observed time of less than 1 second: see **Figure 5.29**.

[10] A single intervention was also chosen so as not to confuse the user or contradict giving a single instruction at level 3.

Chapter 6

User Evaluation

To validate Whitby a user-study was conducted using the application developed. The main goal of the user-study was to assess the effectiveness of the interventions provided by OSWIN. The questions addressed are:

1. How did OSWIN intervene with the system safety analysis?
2. What was the effect of the interventions provided?

It's expected that there will be more "trouble" situations in the log than are intervened on and that can be intervened on in the STPA domain and via Contingent Scaffolding. This is due to the interventions defined not been proven exhaustive, and contingent scaffolding being a pedagogical technique for intervening on a process, rather than teaching or defining the process, which is required for a novice. Contingent Scaffolding has been shown to improve quality of the final product (Daniels 2010), so it's expected to still hold in this case.

The Engineering and Physical Sciences Research Ethics Committee (EPS/FREC) provided approval for the user-study with code: MEEC 20-011.

6.1 Experimental Design

A cohort of users were given the scenario, included in **Appendix D.2**, to undertake an STPA analysis using Whitby. The scenario requests they conduct the analysis for a recommender system, which given information about students and possible course modules can recommend modules for the student to take. Users' logs were captured and users' were observed conducting

the analysis in order to identify "in-trouble" situations. Finally users were questioned to ascertain their expertise with regards to STPA and their experience using the application.

6.1.1 Scenario

The recommender software scenario is chosen as one whose basic architecture will be familiar to the computing students invited to participate: a standard model-view-controller application. It contains data and recommendation-rules gathered from different sources: the model, some rules about how it gathers data and makes recommendations: the controller, and a user-interface: the view.

Furthermore, the recommender system is a suitably simple analysis for a beginner STPA analyst. As a computing system, all actions can be considered as atomic (i.e. having no duration), therefore control actions and potentially unsafe control actions of unexpected duration need not be considered. It's also a small system, with 3-4 controllers involved and 7 control actions, which eases the mental burden on the beginner and is achievable within the time-limit on each user-study.

6.1.2 Participants

The user-study consisted of 37 voluntary participants from the School of Computing, University of Leeds. Participation was offered to any willing student within the School of Computing to ensure some familiarity with the software domain of the scenario. No restrictions were in place with regards to level of study, prerequisite modules, or any other factor besides being a Computing student. The participants were non-expert beginners who are only familiar with the terms, process, and need for STPA. Expert STPA analysts were not invited to participate as the support system is targeted to aid those who are learning STPA. The software also lacks a tutorial feature for teaching STPA to complete beginners, as such a cohort was unavailable this was compensated for with an introductory lecture on STPA analysis including a demonstration analysis.

Each participant chose a time to suit themselves to conduct the study, and were observed individually via video-conferencing software.

6.1.3 Materials and Procedure

User's were provided with participation information, included in **Appendix D.1**, and the recommender scenario included in **Appendix D.2**. The recommender would gather data from students and module leaders such that it could recommend modules to a student. This remit, similar to what could be provided by a party requesting the analysis, included requests that correspond to the losses and hazards to be defined in step 1 in order to limit the scope of the study due to time constraints. It wasn't made explicit that these were sufficient to define the losses and hazards so that participants still had scope to create their own definitions based upon their interpretation of the information, or as in some observed cases, ignore it entirely. Furthermore, in deference to the time constraints, step 4 of the STPA procedure[1] was excluded from the system.

After consenting to take part in the study, a participant is presented with the following sequence of tasks.

1. **Pre-study Expertise Test** This is a questionnaire with 8 questions derived from advice in the STPA Handbook N. Leveson and Thomas 2018 designed to ascertain a base-line of the users knowledge. Information gained here is *not* used to inform Whitby, but only used to compare against the Post-study Expertise Assessment to determine if learning can be demonstrated. The questionnaire is included in **Appendix D.3**.

2. **STPA Analysis** The participant is presented with the application UI and asked to complete an analysis based upon the scenario provided. The OSWIN contingent-scaffolding module is turned on for all participants and will intervene when it can. If a participant is in trouble in some manner that OSWIN cannot intervene to help with, such as a user-interface issue like finding a button to click, then the observer will intervene acting as a tutor. However, on matters of correctness to the scenario, such as asserting an irrelevant loss that's out of scope, no intervention is provided. The distinction is made to aid participants to use the tool, but not aid them in their analysis. **Figure 5.27** and **Figure 5.28** shows the UI in use.

3. **Post-study Expertise Test** This is an exact repeat of the Pre-study Expertise, the same as **Appendix D.3**.

[1] The STPA procedure is outlined in **Appendix A**

6.1.4 Data Collected and Analysis

Each participant is assigned an identifying number so disparate data sources can be joined. These sources are:

- Answers to the STPA test taken pre-study and repeated post-study.
- Log Data, generated by the tool on each action that updates the situation term, including ontology authoring, navigation, intervention, and glossary access actions.
- STAMP model, which is derived from the Log Data
- Observer notes, including any additional intervention given

Each research question is analysed below with a corresponding method of analysis.

6.2 How Did OSWIN Intervene?

This question is to assess the capability to intervene. It's therefore comprised of the following sub-questions:

1. Which of the defined interventions were provided to the users?
2. Were missing or mistake types more prevalent?
3. Were there observed situations where a user was in trouble that OSWIN did not intervene in?
4. Of those observed situations are there any that OSWIN could easily intervene in?

To evaluate these questions both the log data and observer notes are searched for records of interventions.

6.2.1 Use of Defined Interventions

With regard to how many interventions were offered to each user, a histogram is shown in **Figure 6.1**, and the summary statistics are:

- **Total Across All 37 Logs**: 130
- **Mean**: 3.5135
- **Median**: 3
- **Mode**: 4
- **Standard Deviation**: 2.0631

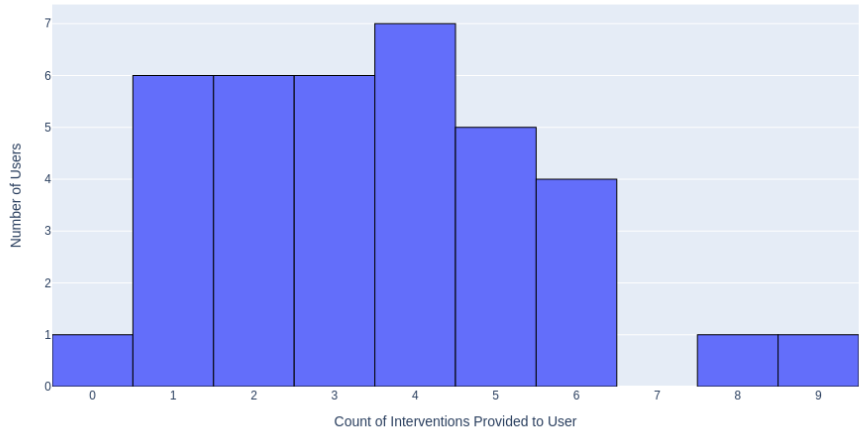

Figure 6.1: Histogram showing distribution of Intervention Count per user

- **Minimum**: 0
- **Maximum**: 9

The users were also provided with a "Request Help" action, which they could use to increase the level of intervention offered. A histogram of these actions is shown in **Figure 6.2**, and the summary statistics are:

- **Total Across All 37 Logs**: 99
- **Mean**: 2.6756
- **Median**: 2
- **Mode**: 2
- **Standard Deviation**: 2.5825
- **Minimum**: 0
- **Maximum**: 10

In total 19 different interventions are defined, 13 of the missing-type and 6 of the mistake-type. In the 37 analyses undertaken a total of 138 interventions were provided. Of the missing-type, 128 interventions were provided, and of the mistake-type 10 were provided. Thus the missing-type interventions were far more prevalent than the mistake type, with only 7 of the 37 users receiving any mistake intervention. **Table 6.1** shows the number of occurrences of each missing intervention, whereas **Table 6.2** shows the same for the mistake-type interventions.

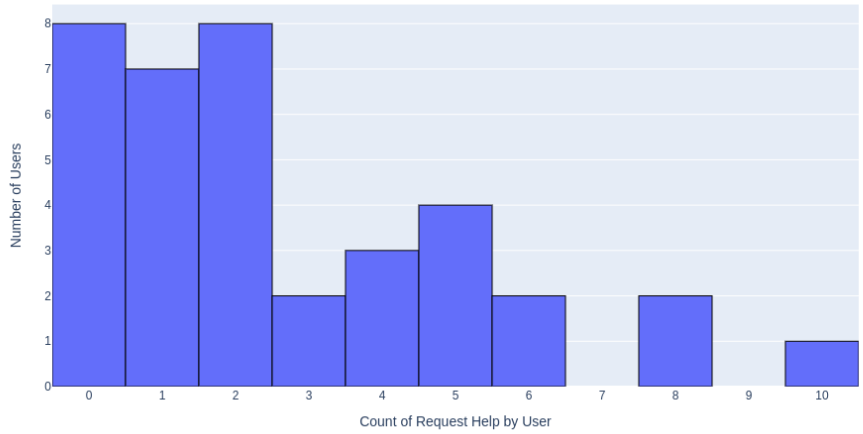

Figure 6.2: Histogram showing distribution of the "Request Help" action count per user

Table 6.1: Counts of missing-type intervention occurrences

Missing Intervention	Count
Hazards will lead to a loss in some worst-case environment	8
Hazards must describe states or conditions to be prevented	6
Check that every controlled physical process is controlled by one or more controllers	18
Check that control actions needed to satisfy the responsibilities are included	26
Check that feedback needed to satisfy the responsibilities is included	23
Ensure traceability is documented to link every unsafe control action with one or more hazards (Providing)	4
Ensure traceability is documented to link every unsafe control action with one or more hazards (Not Providing)	1
Ensure every unsafe control action specifies the context that makes the control action unsafe	8
Ensure the unsafe control action contexts are defined clearly	2
Step 1 precedes Step 2	12
Step 2 precedes Step 3	18
Identify all Providing Potentially Hazardous Control Actions	2
Identify all Not Providing Potentially Hazardous Control Actions	0

Table 6.2: Counts of mistake-type intervention occurrences

Mistake Intervention	Count
Hazards should not include ambiguous or recursive words like "unsafe", "unintended", "accidental", etc.	0
Avoid using ambiguous and vague labels in the control structure: "commands", "feedback", "status", "computer" (Control Action)	0
Avoid using ambiguous and vague labels in the control structure: "commands", "feedback", "status", "computer" (Feedback)	8
If you have more than 7 hazards, consider grouping or combining them to create a more manageable set	0
Fluent can't both hold and not hold in a situation	1
Control Action can't both cause a fluent to hold and not hold	1

From these it can be seen that missing some definition, particularly pertaining to control actions and feedback during step 2 was the most common cause of intervention. Among the mistakes made, the most prevalent was in poor naming of feedback; often users would fail to identify a suitable feedback, such as "student model", "module model", or "recommender accuracy score" and instead used an ambiguous or vague label with some hint at the fluent. For example, user 28 asserted the label: "Algorithm Feedback", user 40 asserted: "check status of student information", and user 55 asserted: "Recommendation Feedback". All of which do not inform a reader about what the feedback is intended to be.

6.2.2 Use of History

Underpinning the contingent scaffolding is the diachronic view of the ontology authoring process, which provides the history of actions and also states via the Situation Calculus reasoner. This is used to determine the level of intervention at which to offer an intervention. If an intervention has been offered previously, resolved, and occurred again, it will be faded by one level from the previous max. So if it were at level 2, it will be offered at level 1. If it were at level 3, it will be offered at level 2.

In post-study analysis fading was found to have occurred on 10 occasions for 7 users. That is

10/130 interventions offered being assigned a level where fading occurred. This was determined by the following query, shown with output. For each user the query conducts the same search and prints out the result. The search it conducts is to divide the users log into three sections: `Early`, `Mid`, and `Later`, with the `Early` section ending with a `request_help` action. This action will increase the intervention level of whatever intervention it was acting upon (the variable `I`). The intervention that was offered is found in the `Early` section, along with the situation query (`Q`) that succeeded. The `Later` section is then checked to ensure that this intervention was offered again. The `Mid` section is when the intervention must have been resolved, this is checked by recreating the situation up until the end of the `Mid` section (`MidEarlySit`) and ensuring that the query (`Q`) does not hold in it. The sorting on the date-time parameter is to remove duplicates found by changing the exact dividing points between the sections in the search process.

```
?- forall(user(U),
  ( bede(U)::sit(Sit), abolish_all_tables,
    sitcalc::situation_list(Sit, SitList),
    findall(I-DT,
      ( append(Later, MidEarly, SitList),
        append(Mid, [request_help(I, U, DT1)|Early], MidEarly),
        once(( member(intervene(I, Q, _, _), Early),
               member(intervene(I, _, _, DT), Later),
               sitcalc::situation_list(MidEarlySit, MidEarly),
               \+ sitcalc::holds(Q, MidEarlySit)))
      ), Cases),
    sort(2, @<, Cases, CaseSet),
    ( CaseSet = []
    -> true
    ; format('---~nUser: ~d~n', U),
      maplist([C, I]>>arg(1, C, I), CaseSet, ISet),
      maplist(writeln, ISet)
    ))).
```

```
User: 14
Check that every controlled physical process is controlled by one \
```

```
    or more controllers
Check that feedback needed to satisfy the responsibilities is included
---
User: 33
Check that control actions needed to satisfy the responsibilities \
    are included
---
User: 36
Check that control actions needed to satisfy the responsibilities \
    are included
Check that feedback needed to satisfy the responsibilities is included
---
User: 40
Avoid using ambiguous and vague labels in the control structure: \
    "commands", "feedback", "status", "computer"
---
User: 41
Check that feedback needed to satisfy the responsibilities is included
---
User: 43
Check that control actions needed to satisfy the responsibilities \
    are included
Check that feedback needed to satisfy the responsibilities is included
---
User: 44
Identify all Providing Potentially Hazardous Control Actions
true.
```

Additionally, there were 8 occasions over 6 users where OSWIN made use of the diachronic view of the ontology to intervene at a level greater than 1. This means that OSWIN made use of the history of actions to inform the level of intervention, determining on these 8 occasions that it was appropriate to begin at level 2 or 3 because the user had previously needed help with this

kind of intervention. This was determined through a search over the logs to find actions where the level of intervention was greater than 1. The query and output follows:

```
?- forall( user(U),
          ( bede(U)::sit(Sit), abolish_all_tables,
            findall(I-N,
              ( sitcalc::member(intervene(I, _, N, _), Sit),
                N > 1
              ), Is),
            ( Is = []
            -> true
            ; format('---~nUser: ~d~n', U),
              maplist(([I-N]>>format('~s @ level: ~d~n', [I, N])), Is))
          )).
---
User: 14
Check that every controlled physical process is controlled by one or \
    more controllers @level: 2
---
User: 23
Check that control actions needed to satisfy the responsibilities \
    are included @level: 3
---
User: 24
Step 2 precedes Step 3 @level: 3
Step 2 precedes Step 3 @level: 2
---
User: 27
Step 2 precedes Step 3 @level: 2
---
User: 36
Check that feedback needed to satisfy the responsibilities is \
    included @level: 3
```

```
Check that feedback needed to satisfy the responsibilities is \
    included @level: 3
---
User: 40
Avoid using ambiguous and vague labels in the control structure: \
    "commands", "feedback", "status", "computer" @level: 2
true.
```

OSWIN has made use of the history, which indicates that this part of the contingent scaffolding framework executes as desired, although the quantity of data gathered is insufficient to draw conclusions regarding the efficacy of using history to determine intervention level on model quality or learning. However, the intention is to demonstrate that the diachronic view of the ontology provides additional, useful information that can be exploited by ontology authoring support tools. OSWIN has demonstrated that explicitly by how fading is accomplished. OSWIN not only reasons on previous actions with regards to the intervention being offered previously and help requested to increase the level, but also can query earlier states of the ontology to determine that the intervention had been resolved in the intermediate time.

6.2.3 Additional Observed Behaviour

In addition to the defined interventions that occurred and were logged, the analyses were also observed, and notes recorded in individual free-text documents, in order to determine if additional "trouble" situations occurred. When a user was in "trouble" with no intervention to be offered, or if they sought help, or offered an opinion of what could help, this was recorded in the documents. The 37 documents were then reviewed to find such notes pertaining to additional support and the common themes abstracted out.

Glossary Improvement

The most common requirement of users pertained to the glossary rather than the interventions, the usage of the glossary histogram is shown in **Figure 6.3**. Users often struggled particularly with understanding "Controlled Process"[2] and asked for definitions beyond the glossary provided:

[2]For example, user-41 requested that the system "be a bit more clear in defining what each (e.g control process vs control action) was", user-39 stated: "Understanding the concept of each role (e.g. the difference between controllers and control processes). When these concepts are confused the system gets harder and harder to navigate/build off of."

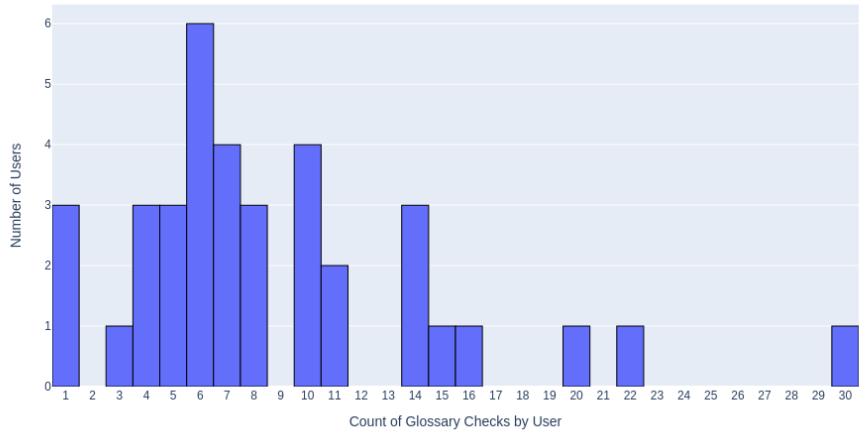

Figure 6.3: Histogram showing the count of the Glossary check action per user

"What activity is the system doing?". The additional explanation provided by the observer was "The thing that the system is doing overall. So if it were a self-driving car it would be driving, for a nuclear power station it would be generating power".

This suggests that the glossary may be able to function within the Contingent Scaffolding framework also, with the escalation of definitions to provide examples and further description if the hint is not understood. Furthermore users requested the glossary functionality also be provided for the predicates as well as the subjects. This contingent-scaffolded-glossary could be incorporated into OSWIN by an additional `request_help` type of action that provides some `definition_of(Term)` parameter for OSWIN to define, although the levels of intervention would require re-examination.

States vs Events

The next most prevalent difficulty noted was in a lack of understanding between situations or states and events. Things that are kinds of events or capabilities were labelled with descriptions of some state, or situation, or fluent in text. Also things that are kinds of situation were labelled with verbs in the text. A couple of users were observed to be struggling because they lacked the fundamental distinction between events and state; they did not yet understand that events change state. For example, user 52's log includes:

```
assert(spo('CA-1',subClassOf,'ControlAction'),
       52,datetime(2022,3,10,16,10,35)).
```

```
assert(spo('CA-1',label,"Incorrect Information about Student"),
        52,datetime(2022,3,10,16,14,46)).
```

A Control Action is a kind of event, but the label they've chosen describes the state of some information about a student. There is no verb in their label. In the inverse direction, user 23's log includes:

```
assert(spo('S-1',subClassOf,'Situation'),
    23,datetime(2022,2,21,10,54,43)).
assert(spo('S-1',label,"User directly gives the system information"),
    23,datetime(2022,2,21,10,55,31)).
```

A situation is an abstraction of state, but the label they've chosen includes the verb "gives", indicating an action.

For those users who had no concept of events and their causal relation to state, additional tuition is required as a prerequisite to system safety analysis[3], which is beyond the scope of a nudge intervention. However, for a user who understands how events and states relate, the use of a verb in some situation label, or the lack of a verb in a capability label could indicate a mistake that OSWIN could identify and nudge on.

A post-study analysis of `Capability` labels (Losses and Control Actions are both subsumed by `Capability` in the STAMP ontology) found that 50% of these labels in the final models and 42% of all 415 capability labels ever asserted contained no verb. This indicates this was a common mistake and a "missing verb in label" kind of intervention could be added to identify when a label is asserted with no verb. It's expected that if a user were to engage with a "missing verb in label" kind of intervention it would improve the quality of their model.

Attribution of Control Actions

The template provided for Unsafe Control Actions in the STPA Handbook (N. Leveson and Thomas 2018) is:

$UCA- < ID >:< Source >< Type >< ControlAction >< Context >< LinktoHazards >$

[3] User-52's feedback requested: "Explanation about the theory of the actions (with examples) which would allow the user to work through their own scenario."

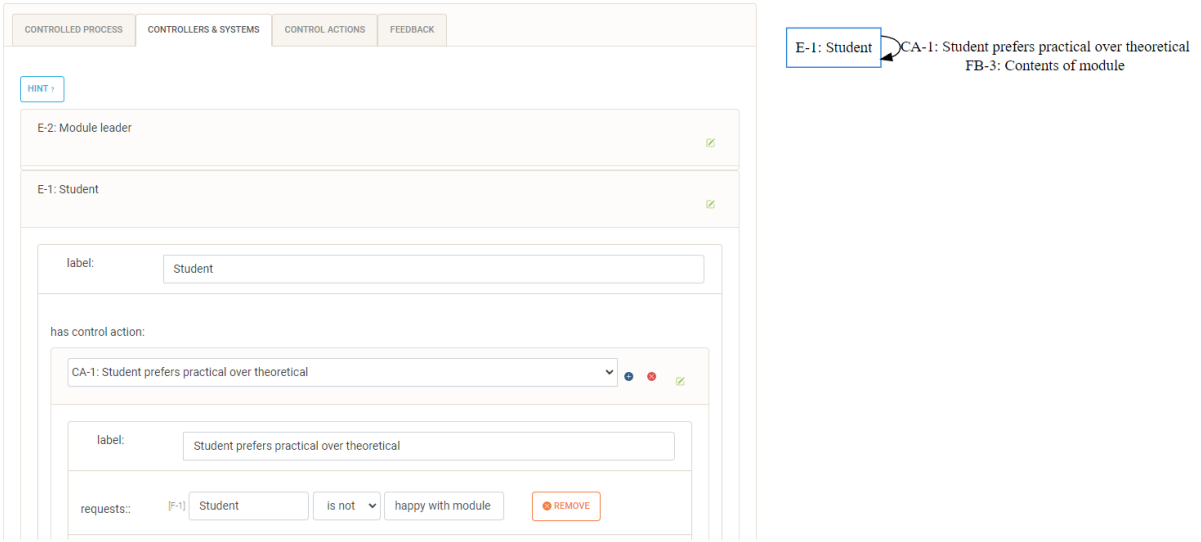

Figure 6.4: An attributed Control Action (CA-1) defined as acting upon self, indicating a mistake from user 48

The $< Source >$ slot is for the Controller who is capable of doing the $< ControlAction >$, which is the 'hasCapability' relationship in the STAMP ontology. So it was expected that the Control Action would be given some label, such as "Add sufficient information about self", and that would be related to the Controller via the provided relation: "Student hasCapability 'Add sufficient information about self'". However, in observation several users attributed the Control Action to the Controller in the label, so it became: "Student hasCapability 'Student Adds sufficient information about self'". An example from user-48 is shown in **Figure 6.4**.

Although this redundant verbosity would be trivial for OSWIN to catch by asserting that Control Action (and Feedback) labels cannot contain a Controller label, it's not clear if this would be beneficial. It was observed for users who defined Control Actions before Controllers that this was their identification of Controllers for inclusion in the model. Furthermore, this should be identifiable through reasoning with the history of authoring actions to determine order of assertions and rudimentary linguistic analysis searching for a noun followed by a verb. Therefore, it should be possible to use this information to determine controllers that may not be defined or other mistakes.

The CA-1 example in **Figure 6.4** contains an additional control action attribution mistake, which should be and can be addressed. Furthermore this mistake was seen in multiple cases. The user has asserted that the `Student` has a capability that requests some fluent to hold that is born by the `Student`. In STAMP terms, they have a controller whose control action effects itself,

which is superfluous and indicative of a mistake. Diagrammatically, the arrow in the diagram is self-referential and it should be pointing to some other box. When constructing the Control Hierarchy diagram self-control is abstracted away in the controllers internal process model, only control between distinct entities should be defined. This mistake can be discovered with the query:

```
logged_assertion(Controller, hasCapability, CA) and
( logged_assertion(CA, requestsToHold, Fluent) or
  logged_assertion(CA, requestsToNotHold, Fluent)
) and
logged_assertion(Fluent, hasBearer, Controller)
```

For user 48 that query would succeed and unify with:

```
logged_assertion('Student', hasCapability, 'CA-1') and
( logged_assertion('CA-1', requestsToHold, 'F-1') or
  logged_assertion('CA-1', requestsToNotHold, Fluent)
) and
logged_assertion('F-1', hasBearer, 'Student')
```

A post-study analysis across all 37 log files found 21 unifications where this query succeeded. In total 12 users could have benefited from this intervention.

This intervention could indicate that a user has missed a controller from their system, and so they're attributing the capability to the fluent bearer in error. It can also indicate, as in this example case, that the control action is poorly defined. However, increasing the level of intervention to an instruction for this query would begin with the assumption that the control action definition is incorrect, it cannot identify the missing controller case.

The attribution of control actions can also indicate two other common mistakes observed during the user study: defining a controller as subject to the controlled process or missing a "hasSubject" relation when some entity has no control actions or feedback. In the already discussed **Figure 5.28** from user 43's analysis, the E-1 entity should have been declared as subject to CP-1: it has no control actions or feedback. This intervention could be caught as a missing type with the query:

```
logged_assertion(E, subClassOf, 'Entity') and
```

```
not ( logged_assertion(E, hasCapability, _) or
        logged_assertion(E, hasFeedback, _)
        ) and
not logged_assertion(CP, hasSubject, E)
```

The case where a user has asserted some controller is subject to a controlled process can be caught with the query:

```
( logged_assertion(C, hasCapability, _) or
  logged_assertion(C, hasFeedback, _)
) and logged assertion(CP, hasSubject, C)
```

A post-study analysis was conducted using these two queries run across all 37 log files and found 143 entities defined. Of these 64 were controllers, and 79 were not controllers. 47 of the 79 non-controllers had not been defined as subject to any controlled process, which is close to 60% of non-controllers, and by 27/37 users. Of the 64 controllers, 14 were defined as subject to some controlled process, or 22% of them, and by 8/37 users. This indicates that implementing these two interventions would benefit the majority of users and aid in improving their model quality.

Adherence to the Scenario

Users were provided with a scenario (included in **Appendix D.2**), which asked them to conduct their STPA analysis on a recommender system that could recommend modules to students. OSWIN cannot account for deviation from the scenario due to its design; OSWIN is not provided with any information about the correct model, instead it only seeks to aid a user make their model correct. This could be observed with users who ignored the provided scenario and immediately set about their own analysis.

The scenario losses are “Decrease of student satisfaction”, and “Decrease of staff satisfaction”[4]. However, user 14 chose “Leaked emails and passwords of users” as their Loss, scored 8% for correctness and 71% for sensibility[5], which means their model was very poor with respect to the scenario provided but above average quality with respect to the scenario they chose. In such a case OSWIN cannot intervene to nudge the user to follow the requested and desired analysis as it doesn’t know what that is by design: it’s a tool to support users’ doing analysis rather

[4]Under "System Safety Goals" in **Appendix D.2**

[5]See the upcoming **Section 6.3.1** for the meaning of these terms

than tutor beginners in how to repeat a pre-formulated analysis. Nothing prohibits this tutoring feature being added at a later date for additional study.

Some less-confident users were seeking reassurance from the observer as to the correctness of their assertions. Although OSWIN cannot determine the correctness to a gold-standard, it could reuse the methodology from BOADiS[6] (Denaux 2013) to provide reassurance by describing what it understands from the assertions, particularly across relations. It can also confirm an intervention has been correctly resolved with a notification when the intervention query first fails, rather than merely vanishing, which often led to the user requesting a further intervention to check their work. Provision of these assurances would require testing to see if they have the intended effect.

6.3 What Effect Did OSWIN Have?

It has been claimed that Contingent Scaffolding will aid a student to produce a better quality product (Daniels 2010) with fewer examples and with better retention of learning (Day and Cordón 1993). In this evaluation only the quality of the product is considered as no examples were provided, and with a single study retention of learning cannot be assessed.

6.3.1 Determining Model Quality

Model quality was determined through marking the final model produced by each user. The mark-scheme was designed to take into account both adherence to the provided scenario (see **Appendix D.2** and for the solution see **Appendix D.4**), dubbed "correctness", and internal consistency, dubbed "sensibility". This distinction is made to aid in distinguishing the quality of a model irregardless of the users adherence to the instruction to complete the analysis for the provided scenario. This distinction was made *a-priori* as OSWIN is designed to aid a user make a high-quality model as opposed to re-asserting some pre-defined one.

The projects were marked along 23 factors. For each factor they received a score between 0 and 3. 3 marks were awarded if all that factor was all present and correct. 2 marks were awarded if there was some missingness or some mistake. 1 mark was awarded if there was some missingness and some mistake. 0 marks were awarded if it was entirely incorrect. If the factor was unattempted then it was left blank to distinguish from incorrect, and will default to 0 when a numerical value is required for the field.

[6] An existing support tool for ontology that provides this kind of support and discussed in **Section 2.4.1**

Of the 23 factors, 8 pertained to correctness:

1. Losses. *Have they defined the correct losses?*
2. Hazards. *Have they defined the correct hazards?*
3. Controlled Process. *Have they defined the correct Controlled Process?*
4. Control Actions: *Have they defined the correct Control Actions?*
5. Feedback: *Have they defined the correct Feedback?*
6. Controllers: *Have they identified the correct Controllers?*
7. Unsafe Control Actions: *Have they correctly identified the Unsafe Control Actions?*
8. UCA Link to Hazards: *Have they correctly linked the UCAs to the correct Hazards?*

Of the 23 factors, the remaining 15 pertain to sensibility:

1. Loss Label: *Does it describe a capability?*
2. Hazard Fluents: *Are they fluents?*
3. Hazard Link to Loss: *Are they sensible given their definitions?*
4. Controlled Process Label: *Is it a Controlled Process?*
5. Controlled Process Subjects: *Are they consistent with their model?*
6. Control Action Label: *Are they Capabilities?*
7. Control Action Requests: *Are they relevant to the hazard conditions?*
8. Control Action Poss: *For UCA, are they defined and sensible?*
9. Feedback Label: *Does it describe some method of recording a fluent?*
10. Feedback records: *Is it relevant to the hazards and recorded by the method?*
11. Controllers Label: *Is it a control system?*
12. Controller Capabilities: *Are they consistent with the model and reasonable?*
13. Controller Feedback: *Are they consistent with the model and reasonable?*
14. Consistent UCA: *Have they identified them consistent with their model?*
15. Consistent UCA Link to Hazards: *Have they identified them consistent with their model?*

Once the factors were marked, three scores were calculated for each user:

- Total %: *Sum marks / Maximum Possible Score*
- Correctness %: *Sum Correctness Factor Marks / Maximum Possible Correctness Score*
- Sensibility %: *Sum Sensibility Factor Marks / Maximum Possible Sensibility Score*

The results of marking are summarised in **Table 6.3**

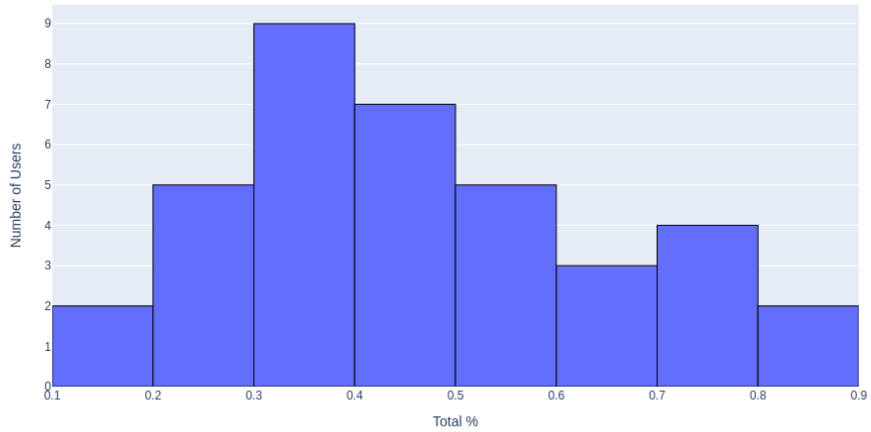

Figure 6.5: Histogram showing distribution of scores for Total %

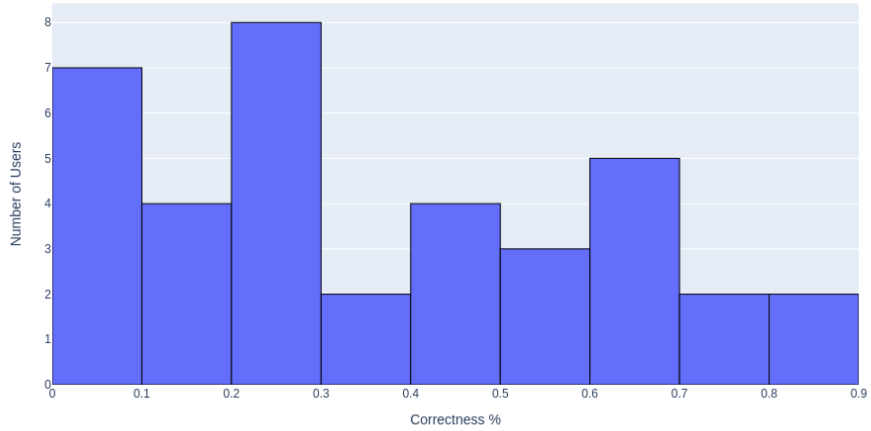

Figure 6.6: Histogram showing distribution of scores for Correctness %

Table 6.3: Summary of Model Quality Scores

Score	Mean	Standard Deviation	Min	Median	Max	Histogram
Total %	0.4756	0.1908	0.1739	0.4499	0.8659	**Figure 6.5**
Correctness %	0.3570	0.2558	0.0000	0.2917	0.8750	**Figure 6.6**
Sensibility %	0.5309	0.1893	0.1778	0.5333	0.9111	**Figure 6.7**

The Total % score averaged 48%. The Sensibility % score was a little higher at 53% and with a slightly lower standard deviation, whereas the Correctness % was lower at 36% and with a broader standard deviation. Thus the users are scoring higher on their sensibility in the

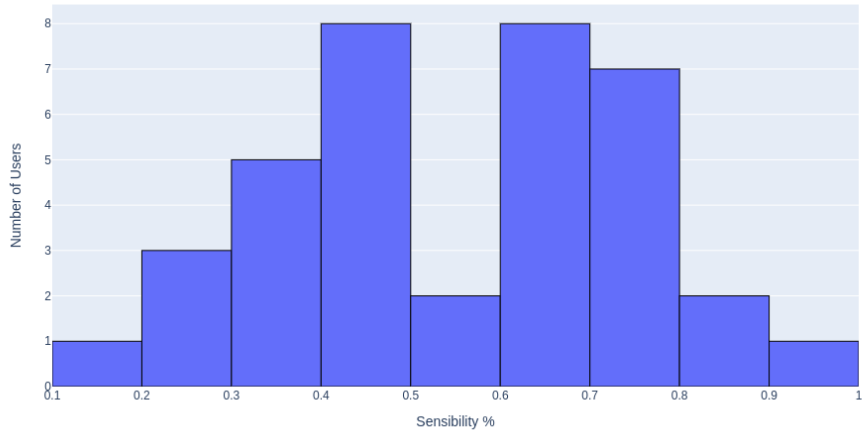

Figure 6.7: Histogram showing distribution of scores for Sensibility %

models than correctness. The users' scores for correctness depend upon both an adherence to the scenario and sensibility: later correct scores will be prohibited by earlier errors. Therefore, along with OSWIN targeting sensibility, it is expected that correctness scores would be lower to some unknown degree.

6.3.2 Determining Learning

To determine if learning occurred during the analysis the STPA pre and post questionnaires were marked and the difference in scores compared. The Wilcoxon Signed Rank Test, for paired data, was used to test for learning based on these scores.

Under this test the null hypothesis is taken as the scores for "STPA Pre" are less than or the same as those of "STPA Post", meaning that the median of the difference (Pre − Post) will either be symmetric around the median at or below 0. The Wilcoxon Signed Rank statistic (sum of the ranks of the differences above zero) is 92.0, however p=0.1914, therefore the null hypothesis cannot be rejected based upon this data and the claim that learning occurred across the population cannot be affirmed.

6.3.3 Do Interventions Effect Task Completion?

Due to the each user-study being assigned a 45 minute slot to complete in the users didn't all have time to complete the study. Furthermore, some users take more individual actions than others. This is important as it's expected that the more work the user did the more interventions they are likely to have encountered, and so it could be argued that some of the

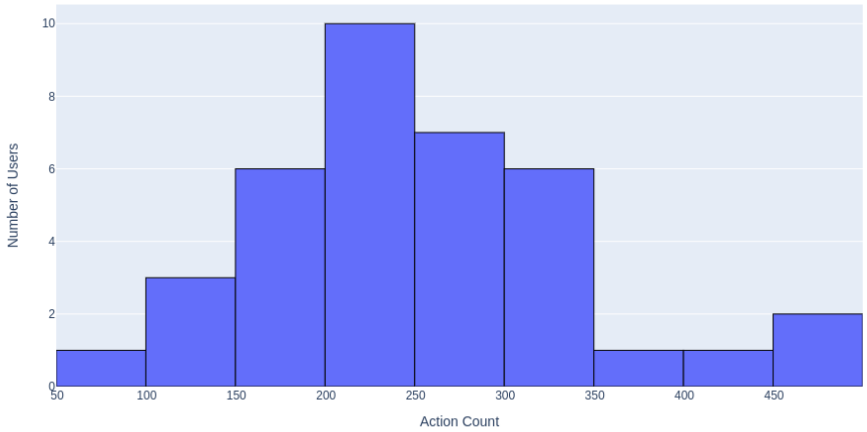

Figure 6.8: Histogram showing distribution of number of actions taken per user

correlation between Intervention Count and any of the model quality scores could be caused by the users who did better taking a greater volume of actions or progressing further, which then triggers more interventions.

For testing correlation Spearman's rank correlation coefficient is used as a non-parametric measure that measures how well the relationship between the two variables can be described by a monotonic and not necessarily linear function.

As an STPA analysis is iterative and definitions within steps can be done in different orders it's not possible to use navigation to determine progress. Instead the 23 factors that are marked can be used to judge what was attempted: a mark of 0, 1, 2, or 3 denotes the factor was attempted and so the mark was assigned. When a factor was not attempted, no mark was assigned and so the missing marks can be used to determine what factors were not attempted. This data was used to generate a measure called "Percent Complete", which describes how much of the system-safety analysis the user had completed. Counting the actions taken is as trivial as counting the length of the log. The summary statistics of these two measures are in **Table 6.4**

Table 6.4: Summary of Work Quantity Measures

Measure	Mean	Standard Deviation	Min	Median	Max	Histogram
Action Count	255.68	86.88	94	238	463	**Figure 6.8**
Percent Complete	0.8166	0.1727	0.3478	0.8696	1	**Figure 6.9**

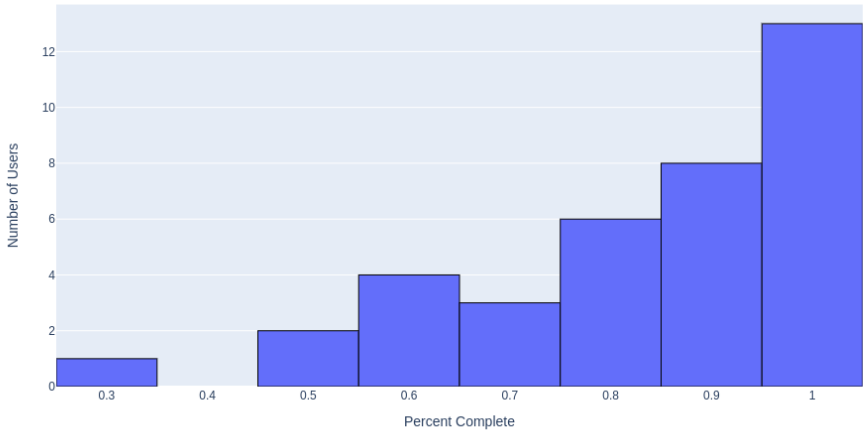

Figure 6.9: Histogram showing distribution of Percent Complete measure

A correlation between these two measures is expected: the more actions a user takes the more they should complete. However, no such correlation was found. With a null hypothesis of a negative or no correlation between the two measures the Spearman Rank Correlation Coefficient was 0.08 with p=0.3.

A correlation between Action Count and Intervention Count would indicate that the more actions taken, the more interventions would be expected, as well as the more interventions provided the more actions would be expected to be taken. A correlation is intuitively expected here, especially as a minimum number of actions are required for certain interventions to trigger. However, if a user has not engaged with OSWIN's interventions then the Intervention Count measure will be 1 or 2, no matter how many actions were taken or were possible due to OSWIN waiting for the intervention already offered being engaged with.

Therefore it's necessary to distinguish between the interventions offered and the interventions engaged with to determine any correlation with Action Count or Percent Complete. It's possible for a user to have read and taken into account an intervention while not resolving it, which would be a form of engagement, but for the purposes of this correlation an engagement is an action that allows an another intervention to be offered, i.e: the intervention is resolved. Therefore an additional variable is measured for this correlation: "Interventions Resolved", where a resolved intervention is one that is offered and whose query no-longer holds in the final situation. The summary statistics compared to Intervention Count are in **Table 6.5**.

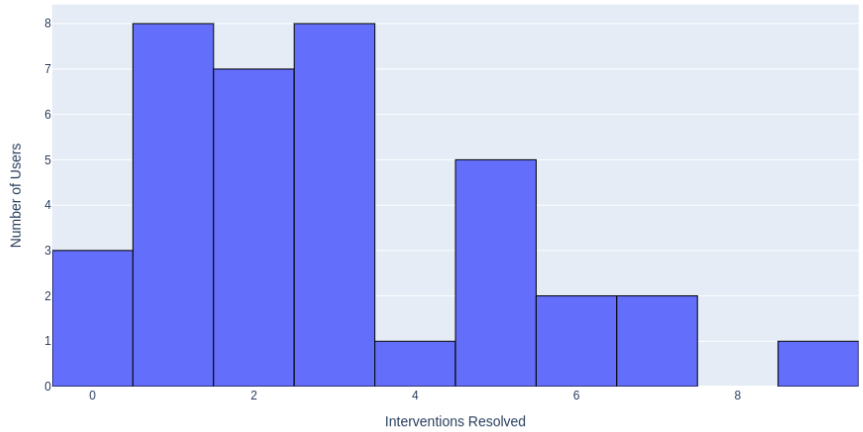

Figure 6.10: Histogram showing distribution of Interventions Resolved per user

Table 6.5: Summary of Intervention Count Measures

Measure	Mean	Standard Deviation	Min	Median	Max	Histogram
Intervention Count	3.51	2.06	0	3	9	**Figure 6.1**
Interventions Resolved	2.32	1.97	0	2	7	**Figure 6.10**

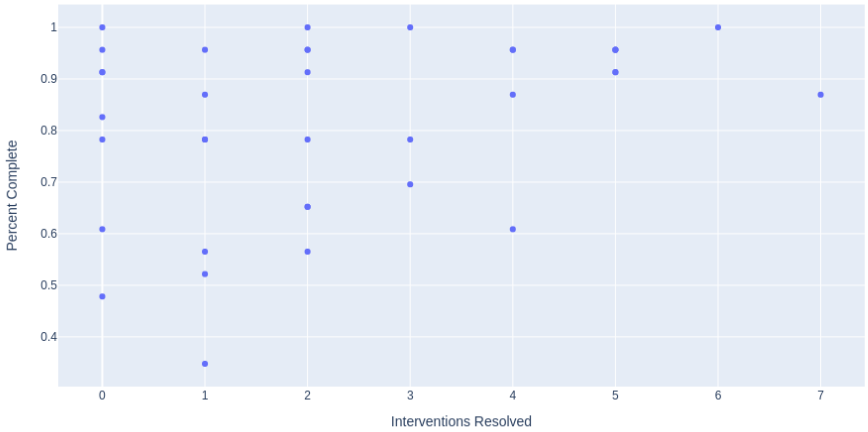

Figure 6.11: Interventions Resolved against Percent Complete including the users excluded from the correlation test who resolved 0 interventions

For users who resolved 0 interventions the number of actions taken cannot correlate with the number of interventions either provided or resolved as the number of interventions was prohibited from increasing. With these 8 users excluded, the Spearman Rank Correlation Coefficients were calculated, in each case the alternate hypothesis is that there is positive correlation between the two variables, shown in **Table 6.6**. Interventions Resolved is plotted against Percent Complete in **Figure 6.11**

Table 6.6: Spearman Rank Correlation between Intervention Count measures and Work Quantity measures

First Variable	Second Variable	R	p
Intervention Count	Action Count	0.3586	0.0280
Interventions Resolved	Action Count	0.3412	0.0351
Intervention Count	Percent Complete	0.4008	0.0156
Interventions Resolved	Percent Complete	0.4626	0.0058

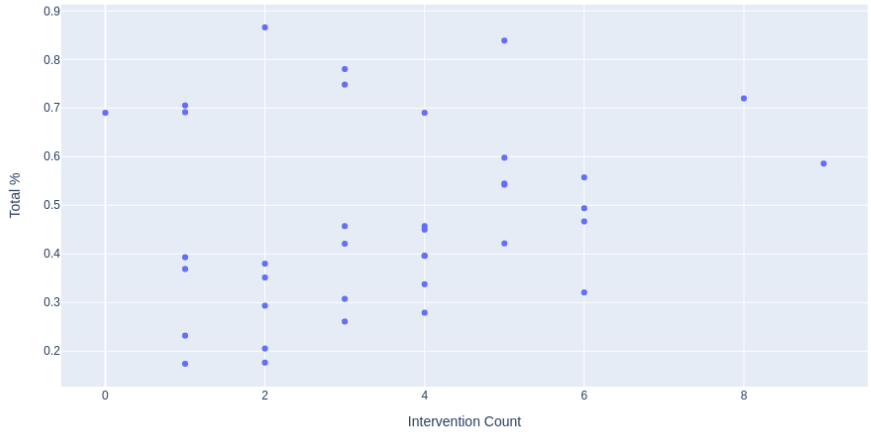

Figure 6.12: Intervention Count plotted against the Total quality mark

In all cases the null hypothesis can be rejected in favour of the alternate: that a positive correlation exists between the First and Second Variables. To resolve an intervention requires taking action and interventions cannot occur without the necessary actions been taken, therefore these correlations are to be expected. However, it's encouraging that the coefficients for Percent Complete are stronger than for Action Count, as there is no correlation between Percent Complete and Action Count (R=0.0890, p=0.3 for single-tailed test with an alternative hypothesis of a positive correlation), and so it appears that the interventions are having a targeted effect on completion rather than just actions, provided the user engages with them.

6.3.4 Does Providing Interventions Positively Effect Model Quality?

If OSWIN's interventions were to positively effect model quality a positive correlation between the number of interventions provided and the quality of the model would be expected. Additionally this correlation should be present for the internal quality (sensibility) and should not be present for the correctness quality mark as OSWIN's interventions cannot effect correctness.

The first test is to see if there is a correlation between "Intervention Count" and the "Total %" quality mark. These two variables are plotted in **Figure 6.12**. Taking the null hypothesis for a single-tailed test as "Intervention Count has no effect or a negative effect on Total %", and alternative hypothesis as "Intervention Count has a positive effect on Total %", the Spearman rank correlation coefficient (R) is 0.28 with p=0.04. Therefore it can be concluded that the Intervention Count weakly correlates with the Total %.

The same is repeated substituting "Total %" for "Correctness %" and "Sensibility %", giving the results shown in **Table 6.7**.

Table 6.7: Spearman Rank Correlation between Intervention Count measures and Model Quality measures

First Variable	Second Variable	R	p
Intervention Count	Total %	0.2801	0.0466
Intervention Count	Correctness %	0.0446	0.3964
Intervention Count	Sensibility %	0.2989	0.0361
Interventions Resolved	Total %	0.3208	0.0264
Interventions Resolved	Correctness %	0.0906	0.2968
Interventions Resolved	Sensibility %	0.3475	0.0175

These results show the stronger correlation for Sensibility than Total, as expected. They also fail to show any correlation for Correctness, as expected. The correlations are also stronger if the interventions are resolved (i.e. the intervention situation query fails in some situation after the intervention is given). **Table 6.6** shows the correlations between Action Count with Intervention Count, and Percent Complete also with Intervention Count. Furthermore, Percent Complete correlates with Total % with $R = 0.68$ and $p = 1.59 \times 10^{-6}$ (single-tailed positive), meaning the more complete a model is the better the model quality. Therefore the correlations taking into account the Percent Complete also need to be examined to determine if the interventions improve model quality regardless of how much of the model they completed.

To account for Percent Complete in the model quality three new measures are introduced based upon the existing ones, which only score based on factors attempted, i.e. with a mark between 0 and 3 (summarised in **Table 6.8**):

- Total Complete %: *Sum marks / Factors attempted count* $\times 3$
- Correctness Complete %: *Sum Correctness Factor Marks / Correctness Factors attempted count* $\times 3$
- Sensibility Complete %: *Sum Sensibility Factor Marks / Sensibility Factors attempted count* $\times 3$

Table 6.8: Summary of Model Quality Scores accounting for completeness

Score	Mean	Standard Deviation	Min	Median	Max
Total Complete %	0.5751	0.1681	0.2807	0.5758	0.8768
Correctness Complete %	0.4070	0.2673	0.0000	0.4166	0.8750
Sensibility Complete %	0.6647	0.1447	0.3333	0.6667	0.9111

The Spearman Rank Correlation Coefficient for these proportional quality measures with the alternative hypotheses being a positive correlation between the First and Second variables are shown in **Table 6.9**.

Table 6.9: Spearman Rank Correlation between Intervention Count measures and proportional Model Quality measures

First Variable	Second Variable	R	p
Intervention Count	Total Complete %	0.0771	0.3251
Intervention Count	Correctness Complete %	-0.028	0.5662
Intervention Count	Sensibility Complete %	0.2090	0.1072
Interventions Resolved	Total Complete %	0.1273	0.2265
Interventions Resolved	Correctness Complete %	0.0155	0.4638
Interventions Resolved	Sensibility Complete %	0.2247	0.0906

No p-value in **Table 6.9** can be used to justify rejecting the null hypotheses that there is no correlation, or a negative correlation between providing interventions and the model quality measures of just the model defined. When the null hypotheses are taken for a two-tailed test as "There is no correlation between First and Second Variables", the p-values also fail to give cause to reject it. From this it cannot be concluded that there's a correlation between the quality of the parts of a model the user has defined and the interventions offered, despite their being correlations between the whole model quality and the number of interventions offered/resolved.

6.3.5 Does Providing Interventions Improve Learning?

Although learning across the entire population could not be claimed, there may be a correlation between learning and intervention count as the questions in the STPA Questionnaire directly

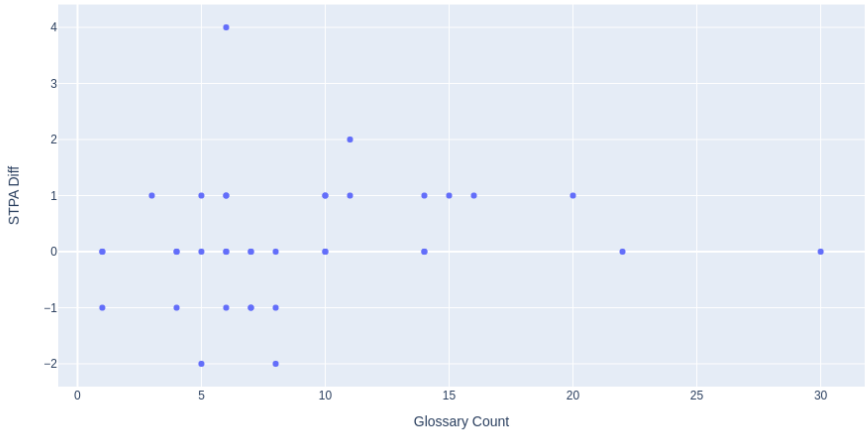

Figure 6.13: Glossary Count plotted against STPA Diff, which is the STPA Post - STPA Pre

relate to the interventions provided. In this case it would be expected that a user who has not learned something in the STPA Questionnaire will be more likely to encounter an intervention related to that learning, which they should learn from.

To test this hypotheses the "STPA Diff" measure is used, which is "STPA Post - STPA Pre". The null hypothesis taken for a single tailed test is "Intervention Count has no effect or a negative effect on STPA Diff", with the alternative hypothesis of "Intervention Count has a positive effect on STPA Diff". For this test the Spearman Rank Correlation Coefficient (R) is -0.06, with p=0.64. Therefore the null hypothesis cannot be rejected and the claim of interventions improving learning cannot be made.

However, interventions were not the sole learning support tool made available in the application. A user was also able to check terms in the glossary. Conducting the same test but substituting "Intervention Count" for "Glossary Count", where "Glossary Count" is the number of times the user checked a glossary term revealed a weak correlation with R = 0.28 and p=0.04. This is plotted in **Figure 6.13**

This is quite a surprising result: the interventions, which the learning test targeted, seem to not have a discernible effect on the learning but the glossary, which depends on the user being pro-active, does. One possible explanation is that the users with most to learn depended more heavily on the glossary in their analyses.

6.4 User Evaluation Conclusions

How did OSWIN intervene with the system safety analysis? It provided 138 interventions with missing-type interventions being more common than mistake-type interventions. Users reported they found the missing type interventions useful, particularly as the model is too large to keep visible at any time and so the interventions aided them in finding what they had missed:

- User-26 stated: "It pointing out missing definitions was useful",
- User-28 with 2 missing and 2 mistake interventions stated: "Hints provided were helpful"
- User-36 stated: "Oswin [sic] provided some good information". Their initial analysis was done with 2 hazards, they then went back to add more, which is when they verbally reported that OSWIN providing missing-interventions became more important to them, helping to find what is missing.
- User-42 stated: "Hint section was quite helpful to find next step."
- User-44 was observed to be relying heavily on the missing interventions, with resolutions to one missing-intervention giving rise to another missing-intervention. Their feedback was to request textual-improvement to make these missing definitions easier to find: "More clear labels when giving help, eg stating what fl-2 related to, a loss or a hazard to make it easier to find"

These users reported that the missing interventions helped them to complete more of their analysis.

OSWIN was also able to make use of the history of actions to: apply fading on 10 occasions, begin from level 1 for the novel interventions, increase the level of intervention as requested and so begin from higher levels for repeated interventions on 8 occasions. This justifies a diachronic view in which the history is available. Furthermore, this history was available to use in the post-study analyses to see how often fading had occurred. The history was also used in analysing potential interventions and additional correlations, for example when conducting the post-study analysis to see if capability labels contained verbs it was possible to check every label asserted whether it had been retracted later or not. This post-study analysis used in conjunction with observation enabled the identification of the multiple interventions for inclusion in **Section 6.2.3**, which would have been relevant to the majority of users and can be included in further-work.

With regards to the effect of the interventions, only correlations can be examined. Correlations

were discovered between interventions offered/resolved and the total/sensibility model quality measures. A correlation was also discovered between Percent Complete and the number of interventions offered/resolved. However no correlation can be claimed for the interventions and model quality when that model quality measurement is only considering the parts that the user defined.

The interventions did not aid, in a statistically significant measurable manner, the user to make higher quality assertions. However they do correlate with the user making a greater quantity of sensible assertions. Given that the ratio of mistake-type to missing-type interventions proffered was 5:64 (4:39 for interventions resolved), this makes some sense: OSWIN primarily offered aid in completing the model over correcting it. Additional potential interventions for common mistakes that would effect multiple users models were identified from the observation, such as the lack of verbs in capability labels, which if implemented may enable OSWIN to be more effective in supporting model quality improvement.

The interventions did nothing to improve adherence to the provided system scenario, as designed, and improved model quality whether the user adhered to the system scenario or not. This is as expected, but in future-work a guided-tutoring feature could be added to walk a beginner through set example analyses if so desired[7]. The same methodology could be used to achieve this by adding additional interventions that pertain solely to the set examples, and adding the necessary GUI features that will guide and instruct the user.

The interventions did not improve learning, however use of the glossary did. Furthermore from the observations of users interacting with the application a strategy for improving the glossary functionality was identified, which would also exploit the contingent scaffolding mechanism.

OSWIN has exploited the diachronic view of the ontology authoring process to offer interventions that correlate with model quality regardless of the scenario, primarily through informing a user about what they were missing, as reported by users. Additional interventions and features have been identified for future-work that should target common mistakes and improve learning.

[7] As requested by user 56 in their feedback: "As first time using STPA, Oswin could have helped in instructing me through the process in more detail, step by step, rather than pointing out large sections that I had missed. Very difficult to do the STPA process without having instructions on how to do the STPA process to hand."

Chapter 7

Conclusions

Can ontology support Scaffolding for System Safety Analysis? This thesis sought to address this question, which means also addressing every domain in that question: ontology, ontology authoring, Contingent Scaffolding, and System Safety Analysis. Furthermore, if ontology can provide support, is that support effective?

The STAMP ontology has been defined, along with formal definitions in Situation Calculus for ontology authoring and Contingent Scaffolding frameworks. These were applied to STPA and incorporated into software (Whitby). To make these frameworks into shareable libraries an architectural constraint was overcome by dependency inversion. Whitby, incorporating the ontology and theoretical frameworks, was then tested in a user-study. The outcomes of the user-study showed promise; the interventions correlated with the completeness as well as the quality of their analyses, and according to user feedback, the missing-type interventions enabled them to complete more of their analysis.

In this chapter the whole research process will be summarised, contributions discussed, generalisability and applicability considered, then directions for future work proposed.

7.1 Synopsis

STAMP was chosen as a model to build upon the existing set-theoretic model from Thomas (2013), with the STPA analysis methodology used before the system is designed due to the additional ontological challenges that representing some conceptualisation imposes. Therefore a STAMP ontology was required. The STAMP model is based in a control systems paradigm with

a focus on safety, therefore ontologies relating to hazards, systems, and behaviour (as in change in state over time abstracted out into conceptual situations for STPA). These informed the authoring of a multi-layered STAMP ontology with modules for control systems and situations.

With the STAMP ontology written, the next goal was to tackle the provision of support. Contingent Scaffolding was chosen due to the claims that it can enable a user to produce a product beyond their capabilities (D. Wood, Bruner, et al. 1976), and it being recommended for use in ill-defined domains (Mitrovic and Weerasinghe 2009), which System Safety Analysis is.

When a human tutor is scaffolding, they make use of their memory of prior events with their tutee to inform how much they'll interfere when they notice the tutee is in trouble. It was noticed that existing ontology authoring support tools, like those discussed in **Section 2.4.1**, did not keep a record of this history. When researching situations for the ontology it was noticed that Situation Calculus can keep a history of actions, and so if an ontology authoring tool could be formalised in Situation Calculus it would make this history available to support a Contingent Scaffolding framework. This is the preposition that the additional information provided by a diachronic view of the ontology over the synchronic view is useful. Therefore, the ontology authoring framework was defined in Situation Calculus.

One of the concerns with Contingent Scaffolding is its lack of flexibility, which can be overcome by constraints. A constraint can be used to recognise when a user has breached that constraint. In the same way, a Situation Calculus query can be used to recognise when a user has entered into a situation that's inadvisable to be in. Therefore, the Situation Calculus definitions were continued to formally define a Contingent Scaffolding framework. To author this framework also required defining an abstract set of levels of interference and determining how being in "trouble" could be both categorised and recognised.

Once the formal frameworks were defined they were applied to STPA prior to being authored in software. Prolog was chosen as the language to write in due to the use of multiple logical formalisms. However, Prolog was found to be lacking the necessary language constructs required to make the code libraries pertaining to the ontology authoring and Contingent Scaffolding frameworks shareable. This was an important factor because both frameworks should be domain agnostic; creating them as third-party libraries supports this claim. Therefore the application was refactored using the Logtalk programming language, which provides the necessary language constructs to do the dependency inversion required. The third-party libraries were extracted

and the application was applied to another domain by Rubinstein Pérez (2021).

Finally a user-study was undertaken to assess the efficacy of the software using the ontology and implementing the ontology authoring and Contingent Scaffolding frameworks. The OSWIN part of the software, responsible for Contingent Scaffolding, was found to be capable of offering relevant interventions in a timely manner, which users found useful according to their feedback. OSWIN primarily offered interventions of the missing-type, and there were correlations between: interventions engaged with and how complete the models were, how complete the models were and model quality, as well as interventions engaged with an model quality. This strongest correlating measure of model quality ignored whether they followed the provided scenario or not, suggesting generalisability across system safety analysis domains.

7.2 Contributions

In this section the significance of the research accomplished will be highlighted on a domain basis.

7.2.1 System Safety Analysis

Ontology for System Safety Analysis has been done before (See **Section 2.2.1**), however the approach to authoring the STAMP ontology is novel for this domain by taking a top-down approach from the very top at a philosophical level of realism and conceptualism, down through theories of situations, to control systems, and finally to the domain of STAMP. This approach addressed some of the issues in the STPA-Sec ontology discussed in **Section 2.2.1**: all terms in the STAMP ontology are defined according to the conceptualisation of their being rather than how they are used in the STAMP model artifacts. The ontology also provides a different perspective on the STAMP model for consideration by the STAMP community. In particular it highlights the different kinds of Safety Constraint, provides a taxonomy of unsafe control actions, and makes clear the abstractions used in the model.

When authoring STAMP ontology a top-ontology was used to discipline the rest of the ontology authoring process (Arp et al. 2015), with the distinction between universal-realism and conceptual being addressed. The approach taken then worked top-down to define modules of situations to capture hypothetical behaviour, and control systems, which is a foundational theory to STAMP (N. Leveson 2017, ch.3). The STAMP module was defined being subsumed by

terms from the higher modules. This resulted in an ontology capable of representing the STAMP model and that enables additional novel reasoning. This is useful to those who create tools to support STAMP analyses, such as Whitby, which exploits only some of the possible reasoning.

The ontology also contributes to the reasoning underlying existing methodologies of support for analysts, which has been a generative approach in the tools discussed in **Section 2.1.3**. These tools will generate all possible states for a user to review if a control action would be unsafe in that state, with some tools also providing rules to filter the states. The ontological representation can also provide this generative support: reasoning with the sets these tools use is discussed in **Appendix B**. However, due to the use of Situation Calculus to inform the `Situation` ontology module, a more targeted approach can be taken to find the states and control actions that will result in a hazard given the information captured in the ontology (also in **Appendix B**). Additionally the ontology contains the means to aid in classification, such as determining if something is an actuator or controller, which could be done with an expert system as discussed in **Section 3.7**. The greater scope of reasoning provided by the ontology provides greater scope for support.

Finally, this thesis proposes a pedagogical approach to providing support in the form of Contingent Scaffolding and demonstrates that it can help non-experts. Given that System Safety Analysis is a high-value and difficult task, the ability to support a non-expert user to produce a higher quality analysis than they are capable of alone is also of high-value.

7.2.2 Ontology Modeling and Authoring

The STAMP ontology when applied with STPA analysis is an unusual use of ontology as it represents an entirely hypothetical world, using conceptual notions of the things in that world, but for a real-world application. The questions asked of the ontology, such as "What would happen if?" and "How can I prevent that from happening?" place requirements on the ontology that are quite different from recording some process of events to recall and reason with. Due to these requirements the additional `hasPositiveEffect` and `hasNegativeEffect` roles were added to the `Situation-Process` behavioural loop to allow reasoning with Situation Calculus with branching time, which may particularly be of interest to users of the Activity Specification Ontology Design Pattern (Katsumi and Fox 2017) and others who need to hypothesise about possible futures.

The Control Systems module in the STAMP ontology provides a contribution to those who are authoring a more complete Control Systems ontology. This module provides a perspective on systems where the main focus is on behaviour and control, rather than structure of the components.

When authoring the ontology *a-priori* modularisation (Thakker et al. 2011) was successfully used to define the architecture. Whilst there is a chapter in this thesis regarding the refactoring of the software, no such refactoring of the ontology was required. This methodology is not a novelty of this thesis, however this is a testimony to the effectiveness of the approach to managing dynamicity and complexity. The methodology was found to be so effective that it was re-applied to the software architecture during the refactor.

This thesis has argued that the additional information in the diachronic view of an ontology is useful for providing support as it retains a history of how the ontology was authored. The diachronic view can not only recall what has been deleted, but it can also recall the complete state of the ontology at any time during the authoring process. Furthermore, additional information about the authoring process can be captured, such as what interventions are offered. The ontology authoring framework is contributed as one means to achieve this diachronic view. Using the ontology authoring framework enabled the Contingent Scaffolding framework to work without the need for an additional user-model to remember past events.

The Contingent Scaffolding framework applied to ontology authoring is also a novel means of supporting an author. As per System Safety Analysis, ontology authoring is an ill-defined task in an ill-defined domain, and so Contingent Scaffolding is recommended as a support methodology for it (Mitrovic and Weerasinghe 2009). This thesis has demonstrated the applicability of Contingent Scaffolding as users who were unaware they were authoring an ontology by extending the STAMP ontology were successfully supported by it.

7.2.3 Intelligent Tutoring

Contingent Scaffolding is applied by tutors, both human and artificially intelligent software, following from its identification and description by D. Wood, Bruner, et al. (1976). This research contributes a formalised model for Contingent Scaffolding such that whether an intervention can be offered or not can be proven, rather than merely tested in software. The formalisation also provides discipline to precise discussion around issues such as what “trouble” means.

This thesis has offered a definition of trouble as some mistake or missing something, derived from the STAMP taxonomy of potentially unsafe control actions in **Section 4.3.2**. It has also proffered a definition of the levels of intervention that is abstract and intended to be generalisable. These were used in the Contingent Scaffolding framework and subsequently applied to STPA and used in Whitby for the user-study.

Furthermore, the formalisation of Contingent Scaffolding was done using Situation Calculus, which provides a means of addressing the flexibility issue that D. Wood and H. Wood (1996a; 1996b) complained of with regards to software implementations of Contingent Scaffolding (See **Section 2.3**). Rather than model what a user should do, this approach to Contingent Scaffolding uses a situation query to determine when the user has made a mistake or missed something. This is similar to the use of constraints in Contingent Scaffolding, where the violation of some constraint is used as an indication that an intervention is needed (Ohlsson 2015), but with expressive Situation Calculus queries, including the over the history of actions, and implemented in a homoiconic language such that the successful query itself can be reasoned with to inform the intervention.

Under the Contingent Scaffolding framework, a user can take a creative approach and interventions will only be offered when one of these situation queries recognises that the user is in a situation that is inadvisable. Furthermore, the libertarian-paternalistic perspective associated with nudges is adopted for the interventions, such that the framework will advise a user but acquiesce should the user dismiss it. This approach was adopted for an ill-defined task where the user may have additional information that the software does not: it enables sensible behaviour for ill-defined tasks but it may not be appropriate for well-defined tasks.

7.2.4 Prolog Programming

The difficulty encountered with the extraction of third-party libraries prior to the refactor of Whitby, which was the motivation to refactor, revealed that the Prolog language lacks the necessary constructs to do dependency inversion. That's not to say that Prolog cannot do dependency inversion, but that first the necessary language constructs need to be defined. These definitions have already been authored as part of the Logtalk programming language, which itself compiles to Prolog.

This thesis then demonstrates the need for dependency inversion to make third-partly libraries,

particularly those that contain rules to be applied to facts in a user's code. It describes the necessary language constructs to achieve dependency inversion and demonstrates it using these constructs in Logtalk. Finally it demonstrates the success of this approach by extracting the frameworks as third-party libraries, some of which are made publicly available[1].

7.3 Generality and Wider Applicability

Abstraction has been a constant theme through the research, with the goal of creating units that could be extracted, reused, and repurposed. Therefore multiple products of the research can be, and have been reapplied.

7.3.1 Reuse of the STAMP Ontology

The STAMP Ontology is reused every time a user extends it when conducting their own STPA analysis. This was accomplished through the `a-priori` modularisation, where the user analysis forms the case-specific layer. However, each module could also be reapplied:

- STAMP module: this thesis has used it for STPA, however it could also be extended for STPASec (akin to the existing ontology discussed in **Section 2.2.1**) and applied to CAST (post-accident analysis).
- Control Systems module: can be applied to anything reasoning about something that can be represented as a control system, which is a very broad domain. It's expected to be particularly useful when reasoning about behaviour, control, and capabilities as the ontology was informed by the perspective of STAMP, which considers these issues.
- Situation module: for reasoning about situations, particularly useful if the intention is to use it in conjunction with a Situation Calculus reasoner to answer questions. As an upper-ontology layer, this module is very general and can be applied to a wide range of domains.
- Top module: as the highest module this is the most general of all. However, there are more established top ontologies available, which have been the subject of far more research too. This module was not authored with the intention of reuse, but only in the interest of independence and because of philosophical differences with BFO and UFO.

[1] Using Logtalk's Package Manager and the "woolpack" Logtalk Pack Registry at `https://github.com/PaulBrownMagic/woolpack`

The STAMP model, and so STAMP ontology shares terms that are found in other System Safety artifacts, such as "Hazard" in Preliminary Hazard Analysis (PHA). This suggests some generalisation could be accomplished such that information garnered from multiple sources could be captured in a single ontology. To accomplish this would require identifying those terms, conducting research over the relevant literature to check that the perspective of those terms in STAMP is valid across the models, extracting them out into a "System Safety" module between the Control Systems module and STAMP module, and using the "System Safety" module to subsume the new methodology specific terms.

7.3.2 Reuse of Ontology Authoring and Contingent Scaffolding Frameworks

The ontology authoring framework was designed to be reused, either by implementing the definition in a programming language of choice, or using the Logtalk pack. It can be used to capture the authoring process of any ontology, and an ontology authoring tool developer can add any additional pertinent action they wish to record.

There is also the possibility to apply this approach to data-capturing processes other than ontology authoring. Event Sourcing is a methodology whereby events are logged which update state, thus state can be recovered by replaying the events (Erb et al. 2016). This shares the logging of the events and determining of state from the events with the Situation Calculus formalisation of the ontology authoring framework. However, because the ontology authoring framework is written in Situation Calculus it's formal, which means state can be proven rather than reconstructed. Furthermore, Situation Calculus enables a greater range of queries as it can be used to reason with the events themselves and needn't restart from the beginning of the log to determine state, but can begin from the end of the log. This similarity suggests the approach taken could be generalised to other applications, such as those using Event Sourcing.

The Contingent Scaffolding framework was also designed for reuse, either by implementing the definition in a programming language of choice, or by using the Logtalk pack as a third-party library. To apply it to another domain requires the definition of missing and mistake interventions, and determining how or if level 4 and level 5 interference can be accomplished based upon if the task is ill-defined and the GUI. The same needs to be done if reapplying it to an ontology authoring task in another domain too, as the interventions are domain and GUI specific.

7.3.3 Reuse of Whitby

Whitby was provided to Rubinstein Pérez (2021), who reapplied it to the robotics planning domain. To do so they needed to create their own ontology for extension, bank of interventions, create their own GUI, and adapt the code that generated interfering instructions at level 4 to work with their GUI. As they had the entire application, complete with web-server, they also needed to change the web-pages served to provide their own GUI HTML and client-side code.

Additionally, the third-party libraries for Situation Calculus reasoning, ontology authoring and Contingent Scaffolding can also be reused without the whole Whitby application. Examples of the reuse of the Situation Calculus library were discussed in **Section 5.7**.

7.4 Future Work

This thesis was an exploration into the merit of using ontology to support System Safety Analysis, as such there is much future work to be undertaken. Therefore the tasks are divided into short-term and long-term projects.

7.4.1 Short-Term Future Work

The user-study revealed interventions that could be added into the intervention bank immediately, and some that would require a little extra work to incorporate linguistic analysis into the software. These interventions, some complete with their situation queries, were discussed in **Section 6.2.3**. In the same **Section 6.2.3**, the idea of an interactive glossary that made use of the Contingent Scaffolding framework to increase in level was proposed. Implementation of this kind of glossary would show versatility of the framework and application to a help-seeking use-case rather than the monitoring which is characteristic of Contingent Scaffolding.

Section 6.2.3 also proposed the implementation of a tutoring feature for instructing based on pre-formulated examples, as well as the application of the methodology from BOADiS (Denaux 2013) to provide reassurance. Although these projects would take a little longer to implement due to changes in GUI and implementation of algorithms, there is little research to undertake to accomplish them and they're expected to resolve particular issues raised during the study.

With these changes in place it would be interesting to conduct another user-study to determine if they have a tangible effect on the learning and analysis quality. Furthermore it would be

interesting to compare results with other cohort targets, such as those who might be undertaking formal study including STPA analysis. For example, trainee engineers who expect to need safety analysis in the future as well as current engineers who are learning STPA for their current roles. Such user-studies would require partnership other Universities who teach STPA as part of their curriculum and industry partners who are using STPA as part of their system-safety toolkit. These user-studies would provide an opportunity to hone the scaffolding to the needs of the more experienced analysts, and unveil additional areas where support could be beneficial.

Due to time constraints on the user-study, step 4 of STPA, which is discussed in **Appendix A**, was excluded from the application. This can be added in immediately with interventions and GUI to support that stage of the analysis.

7.4.2 Long-Term Future Work

There are a few topics that would be interesting to address with the STAMP ontology. Starting from the top module, which is a kind of compromise between BFO and UFO, it would be beneficial to attempt to align it with both of these top ontologies. UFO should be quite achievable given the majority of System Safety related ontologies discussed in **Section 2.2.1** are subsumed by it. Aligning to UFO will impact the lower modules, which must then be revised, for example Fluents would need to take into account qualia.

Aligning with BFO would be more challenging due to the need to first define conceptual terms as concepts, such as System and Situation, although the Modal Relations Ontology (MRO), and other Common Core Ontologies (CCO)[2], which are subsumed by BFO, likely provide some of the solution. Again, the lower modules would likely need some revision to correctly align with BFO too.

Within the STAMP module part of the STAMP ontology, there are temporal terms, such as "Too Soon" and "Too Short", however there is no conceptualisation of time in the ontology, and so what it means to be too soon or too short cannot be captured. Furthermore, a loss denotes that something of value is lost, but there is no ontological representation of that thing of value or value. The STAMP ontology contains little more than the minimum to capture an analysis, but it would be useful to expand outwards from STAMP to incorporate these more peripheral topics. Such expansion could aid in integrating the System Safety Analysis into the

[2]Available at: `https://github.com/CommonCoreOntology/CommonCoreOntologies`

wider system life-cycle if other information were captured in the same ontology such as that pertaining to manufacturing, maintenance, or decommissioning.

Additionally, as System Safety Analysis is often conducted using several methodologies, it would be useful to expand on the STAMP ontology to incorporate additional methodologies, such as FMEA, Change Analysis, or any other listed in **Section 2.1.1**. To accomplish this would involve creating an additional "System Safety" module layer in the ontology as discussed in **Section 7.3.1**.

The additional information the ontology authoring framework can capture has a lot of potential for research in the domain of ontology authoring support. For example, knowing who asserted or retracted what and when could be used to aid collaboration. Another topic, which would require a significant number of logs, would be to search for patterns in behaviour that indicate an author is struggling without the need for pre-defining "trouble" situations. A third would be to generalise the Contingent Scaffolding to an unknown ontology domain.

The ontology authoring and Contingent Scaffolding frameworks in their application to an STPA analysis have been tested in a structured ontology extension process. In Whitby a user is provided with the STAMP ontology to extend and a 3-step process with tailored graphical feedback to guide them through their authoring. However, it should be possible to make use of these frameworks to support an ontology author creating an ontology *ex nihilo*, or extending their chosen top-ontology without a prior known domain with an n-step guided authoring process. The existing tools discussed in **Section 2.4.1** can aid to inform such an application by providing common mistakes (OOPS! (Poveda-Villalón et al. 2014)), the idea of user-asserted "trouble" situations (OntoDebug (Schekotihin et al. 2018)), and guiding the user in their next edits (BOADiS (Denaux 2013)).

In this thesis a decision was made to exploit the reasoning afforded by the ontology to provide scaffolding, however it could have also been exploited to auto-complete assertions, generate definitions, or suggest definitions. A partnership with either an software provider already working with industry partners would provide an opportunity to gather information on how users would like this reasoning to be exploited, and conduct user-studies to compare the relative benefits. Plugin integration into commercial software for analysis would be ideal. This has the greatest potential for the furthest reaching impact and providing a platform to conduct studies across multiple domains, such that the benefits it aims to realize can be gradually enhanced.

Although Whitby was reapplied to the robotics planning domain (Rubinstein Pérez 2021), it would be beneficial to reapply to some other ill-defined task and ill-defined domain. This would greater demonstrate the versatility and domain agnosticism of the software plus underlying frameworks. It would also enable a comparison of user-study results across domains, something Rubinstein Pérez (2021) was prevented from doing due to circumstances beyond their control. Application to a new domain requires a suitable foundational ontology, set of relevant interventions, and new interface.

7.5 And Finally

This research was undertaken to see if ontology could be exploited to support System Safety Analysis. As an initial foray into the topic, this thesis has demostrated that it can. Furthermore, it has brought to light multiple directions this research could be taken in to improve upon the results obtained and to integrate it into the broader system life-cycle. The research also has ramifications for some of its sub-topics, such as ontology authoring support and Contingent Scaffolding in software. Given the value of System Safety, the effect it has on our lives, money, and environment, it has been a priviledge and most worthwhile endeavour to contribute to the enabling of the betterment of the tools that support analysts to do the best job that they can.

References

A. Avižienis, C. Landwehr, Laprie, J., and Randell, B. (Jan. 2004). "Basic Concepts and Taxonomy of Dependable and Secure Computing". In: *IEEE Transactions on Dependable and Secure Computing* 1, pp. 11–33. ISSN: 1545-5971. DOI: 10.1109/TDSC.2004.2. URL: doi.ieeecomputersociety.org/10.1109/TDSC.2004.2.

AAIB (Feb. 2018). *AAIB investigation to Jodel DR1050-M Excellence, G-JODL*. https://www.gov.uk/aaib-reports/aaib-investigation-to-jodel-dr1050-m-excellence-g-jodl.

Ahmad, Jana and Křemen, Petr (2016). "Ontological Anti-patterns in Aviation Safety Event Models". In: *Communications in Computer and Information Science*, pp. 18–30. DOI: 10.1007/978-3-319-45880-9:2.

Anderson, John R (1993). *Rules of the Mind*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Arp, Robert, Smith, Barry, and Spear, Andrew D (2015). *Building Ontologies with Basic Formal Ontology*. Cambridge, MA: The MIT Press. ISBN: 9780262527811.

Artale, Alessandro, Franconi, Enrico, Guarino, Nicola, and Pazzi, Luca (1996). "Part-whole relations in object-centered systems: An overview". In: *Data & Knowledge Engineering* 20.3. Modeling Parts and Wholes, pp. 347–383. ISSN: 0169-023X. DOI: https://doi.org/10.1016/S0169-023X(96)00013-4. URL: http://www.sciencedirect.com/science/article/pii/S0169023X96000134.

Auslander, D. M (1974). *Introducing Systems and Control*. USA: McGraw-Hill.

Aviation Investigation Report: Loss of Rudder in Flight (2007). https://www.skybrary.aero/bookshelf/books/1364.pdf.

Baader, Franz, Horrocks, Ian, Lutz, Carsten, and Sattler, Uli (2017). *An Introduction to Description Logic*. Cambridge, UK: Cambridge University Press.

Bahr, Nicholas J (2015). *System safety engineering and risk assessment*. 2nd ed. Boca Raton: CRC Press.

Barwise, Jon and Perry, John. (1983). *Situations and Attitudes*. eng. Cambridge, Mass.: MIT Press. ISBN: 0262021897.

Bede, Colgrave, Bertram, McClure, Judith, and Collins, Roger (2008). *The ecclesiastical history of the English people*. Oxford University Press.

Beer, Stafford (1979). *The heart of enterprise*. Bath: Wiley, p. 118.

Bennett, Brandon and Galton, Antony P. (2004). "A unifying semantics for time and events". In: *Artificial Intelligence* 153.1-2, pp. 13–48. DOI: 10.1016/j.artint.2003.02.001.

Björnsdóttir, Svana Helen and Rejzek, Martin (2017). "Embedding STPA into a highly successful risk management software application". In: *6th MIT STAMP Workshop, Boston, USA, 27-30 March 2017*. ZHAW Zürcher Hochschule für Angewandte Wissenschaften.

Borgo, Stefano, Carrara, Massimiliano, Garbacz, Pawel, and Vermaas, Pieter E. (2009). "A formal ontological perspective on the behaviors and functions of technical artifacts". In: *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 23.01, p. 3. DOI: 10.1017/s0890060409000079.

Brown, Paul S, Cohn, Anthony G, Hart, Glen, and Dimitrova, Vania (2020). "Contingent Scaffolding for System Safety Analysis". In: *International Conference on Artificial Intelligence in Education*. Springer, pp. 395–399.

Brown, Paul S, Dimitrova, Vania, Hart, Glen, Cohn, Anthony G, and Moura, Paulo (2021). "Refactoring the Whitby Intelligent Tutoring System for Clean Architecture". In: *Theory and Practice of Logic Programming* 21.6, pp. 818–834.

Brown, Paul S. (Apr. 2022). *PaulBrownMagic/STAMP-Ontology: Initial Release*. Version v1.0.0. DOI: 10.5281/zenodo.6489774. URL: https://doi.org/10.5281/zenodo.6489774.

Daniels, Harry (2010). *Vygotsky and Pedagogy*. London: Routledge.

Day, Jeanne D. and Cordón, Luis A. (1993). "Static and Dynamic Measures of Ability: An Experimental Comparison." In: *Journal of Educational Psychology* 85.1, pp. 75–82. DOI: 10.1037/0022-0663.85.1.75.

Del Frate, Luca (2014). "Failure: Analysis of an Engineering Concept". PhD thesis. Delft University of Technology.

Denaux, Ronald (2013). "Intuitive Ontology Authoring using Controlled Natural Language". PhD thesis. University of Leeds.

Dorf, Richard C and Bishop, Robert H (2011). *Modern control systems*. 12th ed. Boston [Mass.]: Pearson.

Dowson, Mark (1997). "The Ariane 5 software failure". In: *ACM SIGSOFT Software Engineering Notes* 22.2, p. 84.

Du Boulay, Benedict and Luckin, Rosemary (Jan. 2001). "Modelling Human Teaching Tactics and Strategies for Tutoring Systems". In: *International Journal of Artificial Intelligence in Education* 12, pp. 235–256. DOI: 10.1007/s40593-015-0053-0.

Erb, Benjamin, Habiger, Gerhard, and Hauck, Franz J (2016). "On the potential of event sourcing for retroactive actor-based programming". In: *First Workshop on Programming Models and Languages for Distributed Computing*, pp. 1–5.

Fleming, Cody Harrison, Spencer, Melissa, Thomas, John, Leveson, Nancy, and Wilkinson, Chris (2013). "Safety assurance in NextGen and complex transportation systems". In: *Safety Science* 55, pp. 173–187. DOI: 10.1016/j.ssci.2012.12.005.

Gamma, Erich, Helm, Richard, Johnson, Ralph, and Vlissides, John (1997). *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, Massachusetts: Addison Wesley.

Gangemi, Aldo and Mika, Peter (2003). "Understanding the Semantic Web through Descriptions and Situations". In: *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*. Ed. by Robert Meersman, Zahir Tari, and Douglas C. Schmidt. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 689–706. ISBN: 978-3-540-39964-3.

Group, W3C SSN Incubator (2005). *SSN Ontology example: Wind Sensor*. https://www.w3.org/2005/Incubator/ssn/ssnx/meteo/WM30.

Gruber, Thomas R. (1995). "Toward principles for the design of ontologies used for knowledge sharing?" In: *International Journal of Human-Computer Studies* 43.5-6, pp. 907–928. DOI: 10.1006/ijhc.1995.1081.

Guizzardi, Giancarlo, Botti Benevides, Alessander, Fonseca, Claudenir M, Porello, Daniele, Almeida, João Paulo A, and Prince Sales, Tiago (2021). "UFO: Unified Foundational Ontology". In: *Applied Ontology* Preprint, pp. 1–44.

Guizzardi, Giancarlo and Wagner, Gerd (2011). "Towards an Ontological Foundation of Discrete Event Simulation". In: *Proceedings of the 2010 Winter Simulation Conference (WSC)*, pp. 652–664.

Gurgel, D. L., Hirata, C. M., and M. Bezerra, J. d. (Sept. 2015). "A rule-based approach for safety analysis using STAMP/STPA". In: *2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC)*, 7B2-1-7B2–8. DOI: 10.1109/DASC.2015.7311464.

Haller, Armin, Janowicz, Krzysztof, Cox, Simon, Le Phuoc, Danh, Taylor, Kerry, and Lefrançois, Maxime (2017). *Semantic Sensor Network Ontology*. https://www.w3.org/TR/2017/REC-vocab-ssn-20171019/.

Hayes, Patrick J. and Patel-Schneider, Peter F. (2020). *RDF 1.1 Semantics*. URL: https://www.w3.org/TR/2014/REC-rdf11-mt-20140225/%5C#reification (visited on 05/01/2020).

Herre, Heinrich (2010). "General Formal Ontology (GFO): A foundational ontology for conceptual modelling". In: *Theory and applications of ontology: computer applications*. Springer, pp. 297–345.

INBAS (2018). *Ontologie pro bezpecnost v letectvi dokumentace*. https://dev.inbas.cz/owldoc/index.html.

ISO/IEC (1995). *International Standard ISO/IEC 13211-1 Information Technology — Programming Languages — Prolog — Part I: General core*. ISO/IEC.

– (2000). *International Standard ISO/IEC 13211-2 Information Technology — Programming Languages — Prolog — Part II: Modules*. ISO/IEC.

Jenkins, Steven (2018). *Semantic Technologies for Systems Engineering*. `https://github.com/st4se`. Accessed: 2018-3-6.

Katsumi, Megan and Fox, Mark (2017). "Defining Activity Specifications in OWL". In: *Proceedings of the Workshop on Ontology Patterns*.

Katz, Sandra and Lesgold, Alan (1993). "The Role of the Tutor in Computer-Based Collaborative Learning Situations". In: *Computers as Cognitive Tools*. Ed. by Susanne P. Lajoie and Sharon J. Derry. Hillsdale, NJ: Lawrence Erlbaum Associates, pp. 289–318.

Kokar, Mieczyslaw M, Matheus, Christopher J, and Baclawski, Kenneth (2009). "Ontology-based situation awareness". In: *Information fusion* 10.1, pp. 83–98.

Kostov, Bogdan, Ahmad, Jana, and Křemen, Petr (2017). "Towards Ontology-Based Safety Information Management in the Aviation Industry". In: *On the Move to Meaningful Internet Systems: OTM 2016 Workshops*. Cham: Springer International Publishing, pp. 242–251. ISBN: 978-3-319-55961-2.

Kowalski, R and Sergot, M (Jan. 1986). "A Logic-based Calculus of Events". In: *New Gen. Comput.* 4.1, pp. 67–95. ISSN: 0288-3635. DOI: `10.1007/BF03037383`. URL: `http://dx.doi.org/10.1007/BF03037383`.

Lacy, Lee W. (2005). *OWL: Representing Information Using The Web Ontology Language*. Victoria: Trafford.

Leveson, Nancy (2004). "A new accident model for engineering safer systems". In: *Safety science* 42.4, pp. 237–270.

– (2017). *Engineering a safer world*. Cambridge, Mass.: The MIT Press.

Leveson, Nancy and Thomas, John (2018). *STPA Handbook*. `https://psas.scripts.mit.edu/home/get:file.php?name=STPA:handbook.pdf`.

Leveson, Nancy G and Turner, Clark S (1993). "An investigation of the Therac-25 accidents". In: *Computer* 26.7, pp. 18–41.

Liddell, H. G. and Scott, R. (1940). *A Greek-English Lexicon*. URL: http://www.perseus.tufts.edu/hopper/text?doc=Perseus%3Atext%3A1999.04.0057%3Aentry%3Dsu%2Fsthma (visited on 12/12/2017).

Lynch, Collin, Ashley, Kevin D, Pinkwart, Niels, and Aleven, Vincent (2009). "Concepts, structures, and goals: Redefining ill-definedness". In: *International Journal of AI in Education* 19.3, pp. 253–266.

Martin, Robert C (2018). *Clean Architecture: A Craftman's Guide to Software Structure and Design*. Hudson, New Jersey: Prentice Hall.

Masolo, Claudio, Borgo, Stefano, Gangemi, Aldo, Guarino, Nicola, and Oltramari, Alessandro (2002). "Wonderweb deliverable d17". In: *Science Direct Working Paper No S1574-034X (04)*, pp. 70214–8.

– (2003). *WonderWeb Deliverable D18 Ontology Library (final)*. Tech. rep. IST Project 2001-33052 WonderWeb: Ontology Infrastructure for the Semantic Web.

McCarthy, John and Hayes, Patrick J. (1969). "Some Philosophical Problems from the Standpoint of Artificial Intelligence". In: *Machine Intelligence 4*. Ed. by B. Meltzer and D. Michie. reprinted in McC90. Edinburgh University Press, pp. 463–502.

Merrill, Douglas C., Reiser, Brian J., Ranney, Michael, and Trafton, J. Gregory (1992). "Effective Tutoring Techniques: A Comparison of Human Tutors and Intelligent Tutoring Systems". In: *Journal of the Learning Sciences* 2.3, pp. 277–305. DOI: 10.1207/s15327809jls0203:2.

Mitrovic, Antonija and Weerasinghe, Amali (2009). "Revisiting ill-definedness and the consequences for ITSs". In: *Artificial intelligence in education: Building learning systems that care from knowledge representation to affective modelling*, pp. 375–382.

Moura, Paulo (Apr. 2011). "Programming Patterns for Logtalk Parametric Objects". In: *Applications of Declarative Programming and Knowledge Management*. Ed. by Salvador Abreu

and Dietmar Seipel. Vol. 6547. Lecture Notes in Artificial Intelligence. Berlin Heidelberg: Springer-Verlag, pp. 52–69.

– (May 2021). *The Logtalk Handbook*. Release 3.46.0.

Mueller, Erik T (2015). *Commonsense reasoning*. 2nd ed. Waltham, MA: Elsevier.

Nguyen, Vinh, Bodenreider, Olivier, and Sheth, Amit (2014). "Don't like RDF reification?" In: *Proceedings of the 23rd international conference on World wide web - WWW '14*. DOI: 10.1145/2566486.2567973.

Noy, N. and McGuinness, Deborah (Jan. 2001). "Ontology Development 101: A Guide to Creating Your First Ontology". In: *Knowledge Systems Laboratory* 32.

Nuseibeh, Bashar (1997). "Ariane 5: who dunnit?" In: *IEEE Software* 14.3, p. 15.

O'Keefe, Richard A. (1990). *The Craft of Prolog*. Cambridge, Massachusetts: The MIT Press.

Odell, James J. (Jan. 1994). "Six Different Kinds Of Composition". In: *Journal Of Object-Oriented Programming* 5.8.

Ohlsson, Stellan (2015). "Constraint-Based Modeling: From Cognitive Theory to Computer Tutoring – and Back Again". In: *International Journal of Artificial Intelligence in Education* 26.1, pp. 457–473.

Pawlicki, Todd, Samost, Aubrey, Brown, Derek W., Manger, Ryan P., Kim, Gwe-Ya, and Leveson, Nancy G. (2016). "Application of systems and control theory-based hazard analysis to radiation oncology". In: *Medical Physics* 43.3, pp. 1514–1530. DOI: 10.1118/1.4942384.

Pereira, Daniel Patrick, Hirata, Celso, and Nadjm-Tehrani, Simin (2019). "A STAMP-based ontology approach to support safety and security analyses". In: *Journal of Information Security and Applications* 47, pp. 302–319.

Poveda-Villalón, María, Gómez-Pérez, Asunción, and Suárez-Figueroa, Mari Carmen (2014). "OOPS! (OntOlogy Pitfall Scanner!)" In: *International Journal on Semantic Web and Information Systems* 10.2, pp. 7–34. DOI: 10.4018/ijswis.2014040102.

Reiter, Raymond (2001). *Knowledge in Action*. Cambridge, Massachusetts: The MIT Press.

Rising, John M. and Leveson, Nancy G. (2018). "Systems-Theoretic Process Analysis of space launch vehicles". In: *Journal of Space Safety Engineering* 5.3-4, pp. 153–183. DOI: 10.1016/j.jsse.2018.06.004.

Rubinstein Pérez, Federico (2021). "Desarrollo de un Sistema de Gestión del Conocimiento que utiliza Contingent Scaffolding para asistir a un operador industrial en la definición de procesos de un cobot". B.S. thesis. Universitat Politècnica de Catalunya.

Schekotihin, Konstantin, Rodler, Patrick, and Schmid, Wolfgang (2018). "OntoDebug: Interactive Ontology Debugging Plug-in for Protégé". In: *Foundations of Information and Knowledge Systems*. Ed. by Flavio Ferrarotti and Stefan Woltran. Cham: Springer International Publishing, pp. 340–359.

Shanahan, Murray (1997). *Solving the Frame Problem*. Cambridge, Massachusetts: The MIT Press.

Shepardson, David (2018). *2017 safest year on record for commercial passenger air travel - groups*. https://uk.reuters.com/article/uk-aviation-safety/2017-safest-year-on-record-for-commercial-passenger-air-travel-groups-idUKKBN1EQ17F.

Simons, Peter (1987). *Parts. A study in Ontology*. Oxford: Clarendon Press.

Souza, Fellipe GR, Pereira, Daniel P, Pagliares, Rodrigo M, Nadjm-Tehrani, Simin, and Hirata, Celso M (2019). "WebSTAMP: a web application for STPA & STPA-Sec". In: *MATEC Web of Conferences*. Vol. 273. EDP Sciences, p. 02010.

Stephans, R.A. (2012). *System Safety for the 21st Century: The Updated and Revised Edition of System Safety 2000*. New Jersey: Wiley.

Stephenson, Arthur (1999). *Mars Climate Orbiter Mishap Investigation Board Phase I Report*. Mars Climate Orbiter Mishap Investigation Board.

Thakker, Dhavalkumar, Dimitrova, Vania, Lau, Lydia, Denaux, Ronald, Karanasios, Stan, and Yang-Turner, Fan (2011). "A Priori Ontology Modularisation in Ill-Defined Domains". In: *Proceedings of the 7th International Conference on Semantic Systems*. I-Semantics '11. Graz,

Austria: Association for Computing Machinery, pp. 167–170. ISBN: 9781450306218. DOI: 10.1145/2063518.2063541. URL: https://doi.org/10.1145/2063518.2063541.

Thaler, Richard H and Sunstein, Cass R (2009). *nudge: Improving decisions about health, wealth and happiness*. Penguin Books Ltd.

Thomas, John (2013). "Extending and Automating a Systems-Theoretic Hazard Analysis for Requirements Generation and Analysis". PhD thesis. Massachusetts Institute of Technology.

Vargas, Abigail Parisaca and Bloomfield, Robin (2015). "Using Ontologies to Support Model-based Exploration of the Dependencies between Causes and Consequences of Hazards." In: *KEOD*, pp. 316–327.

Vesely, W. E, Roberts, N., Goldberg, F., and Haasl, D. (1981). *Fault tree handbook*. Washington, D.C: U.S. Nuclear Regulatory Commission.

Vieu, Laure (Jan. 2006). "On the transitivity of functional parthood". In: *Applied Ontology* 1, pp. 147–155.

Vieu, Laure and Aurnague, Michel (2007). "Part-of relations, functionality and dependence". In: *The Categorization of Spatial Entities in Language and Cognition*, pp. 307–336. DOI: 10.1075/hcp.20.18vie.

Vincoli, Jeffery W (2006). *Basic Guide To System Safety*. 2nd ed. New Jersey: Wiley.

Wilson, B (1984). *Systems: concepts, methodologies and applications*. Wiley.

Winston, Morton E., Chaffin, Roger, and Herrmann, Douglas (1987). "A Taxonomy of Part-Whole Relations". In: *Cognitive Science* 11.4, pp. 417–444. DOI: 10.1207/s15516709cog1104:2.

Wood, David (1998). *How Children Think and Learn*. Oxford, UK: Blackwell.

– (2018). "Commentary: Contribution of Scaffolding to Learning and Teaching: Interdisciplinary Perspectives". In: *International Journal of Educational Research* 90, pp. 248–251. DOI: 10.1016/j.ijer.2018.03.005.

Wood, David, Bruner, Jerome S., and Ross, Gail (1976). "The Role of Tutoring in Problem Solving". In: *Journal of Child Psychology and Psychiatry* 17.2, pp. 89–100. DOI: 10.1111/j.1469-7610.1976.tb00381.x.

Wood, David and Wood, Heather (1996a). "Contingency in Tutoring and Learning". In: *Learning and Instruction* 6.4, pp. 391–397. DOI: 10.1016/s0959-4752(96)00023-0.

– (Mar. 1996b). "Vygotsky, Tutoring and Learning". In: *Oxford Review of Education* 22, pp. 5–16. DOI: 10.1080/0305498960220101.

Wood, David, Zaidman, Marsha, Ruth, Luke, and Hausenblas, Michael (2014). *Linked Data. Structured data on the Web*. New York: Manning publications co.

Wood, Heather and Wood, David (1999). "Help seeking, learning and contingent tutoring". In: *Computers & Education* 33.2-3, pp. 153–169. DOI: 10.1016/s0360-1315(99)00030-5.

Zeigler, Bernard P (1976). *Theory of modelling and simulation*. New York: Wiley.

Zhou, Jiale, Hänninen, Kaj, and Lundqvist, Kristina (2017). "A Hazard Modeling Language for Safety-Critical Systems Based on the Hazard Ontology". In: *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications*, pp. 301–304.

Zhou, Jiale, Hänninen, Kaj, Lundqvist, Kristina, and Provenzano, Luciana (2017). "An ontological approach to identify the causes of hazards for safety-critical systems". In: *2017 2nd International Conference on System Reliability and Safety (ICSRS)*, pp. 405–413.

Appendix A

Introduction to STPA

In this section a brief overview of the steps taken by an STPA analyst is provided. For a more detailed explanation see the STPA Handbook (N. Leveson and Thomas 2018), from which this Appendix is sourced. STPA is a 4-Step process, as depicted in **Figure A.1**.

A.1 Step 1: Define Purpose of the Analysis

The purpose of this step is to define the scope of the analysis. To do this the losses, hazards, and safety constraints must be defined.

The template for a loss definition is:

⟨ID⟩ : ⟨Description⟩

Common examples include:

- L-1: Loss of life or injury to people

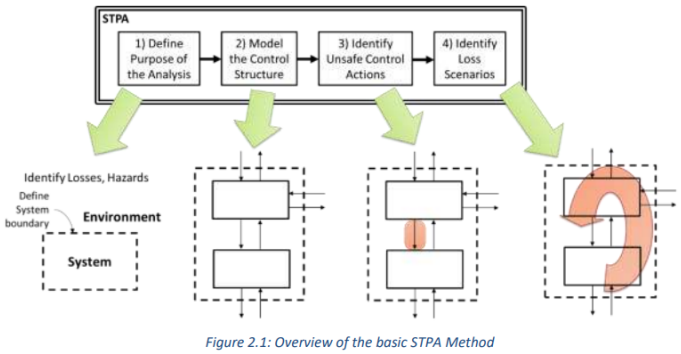

Figure A.1: The 4 steps of STPA

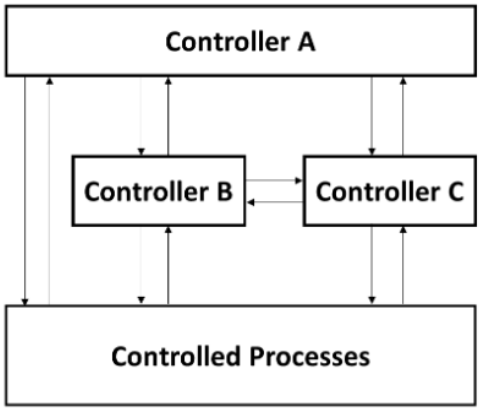

Figure A.2: Control Hierarchy Diagram Template

- L-2: Loss of or damage to vehicle
- L-3: Loss of customer satisfaction

The template for a hazard definition is:

⟨ID⟩ : ⟨System⟩&⟨Unsafe Condition⟩&⟨Link to Losses⟩

For example:

- H-1: Door open and power on [L-1]

The templates for safety constraint are:

⟨ID⟩ : ⟨System⟩&⟨Condition to Enforce⟩&⟨Link to Hazards⟩

⟨ID⟩ : If ⟨hazard⟩ occurs, then ⟨what needs to be done to prevent or minimize a loss⟩

For example:

- SC-1: Power off and door open [H-1]
- SC-2: If H-1 occurs, then turn off power

A.2 Step 2: Model the Control Structure

The purpose of this step is to define the control structure so that it may be analysed for safety. This is done diagrammatically using the template in **Figure A.2**. The analysts define the Controllers, which are the entities in the system capable of deciding which control actions to do, taking into account feedback. The control actions are captured by downward arrows, feedback by upward arrows. The control actions and feedback are shown in **Figure A.3**, which is an example for a tea-making machine.

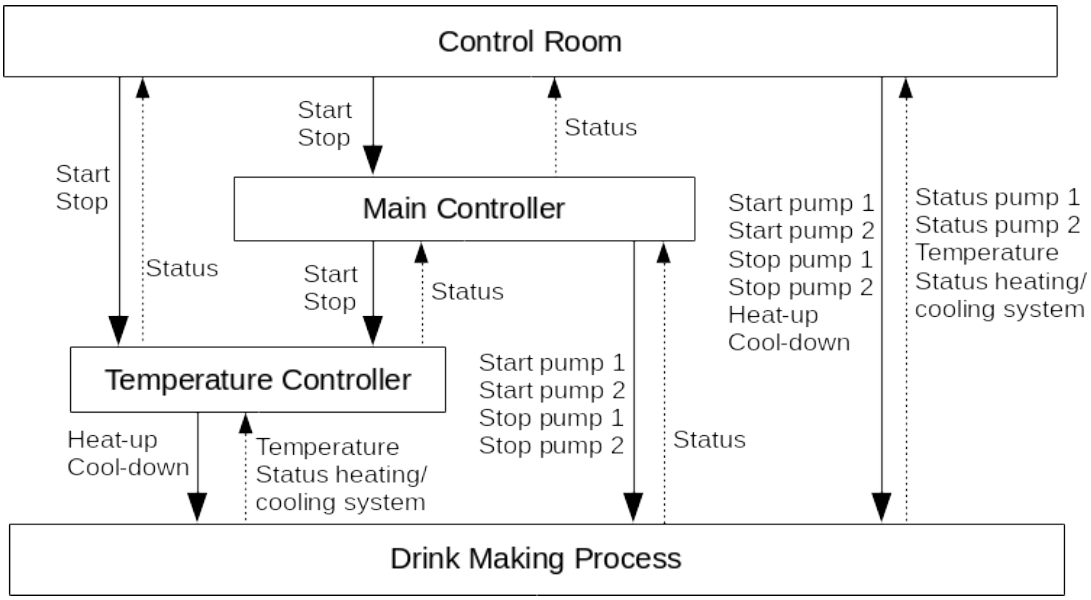

Figure A.3: Control Hierarchy Diagram Example for a tea-making machine

A.3 Step 3: Identify Unsafe Control Actions

In this step the control actions are considered on a case-by-case basis to determine if providing them, not providing them, or providing them in a temporally different manner could be unsafe.

The template for an unsafe control action is:

⟨ID⟩ : ⟨Source⟩ ⟨Type⟩ ⟨Control Action⟩ ⟨Context⟩ ⟨Link to Hazards⟩

For example:

- UCA-1: Power Controller Provides Power On when Door Open [H-1]

A.4 Step 4: Identify Loss Scenarios

In the final step loss scenarios are described, which are descriptions of the causal factors that could lead to the unsafe control actions and hazards. This is done using a diagram to guide the thought process, which is shown in **Figure A.4**.

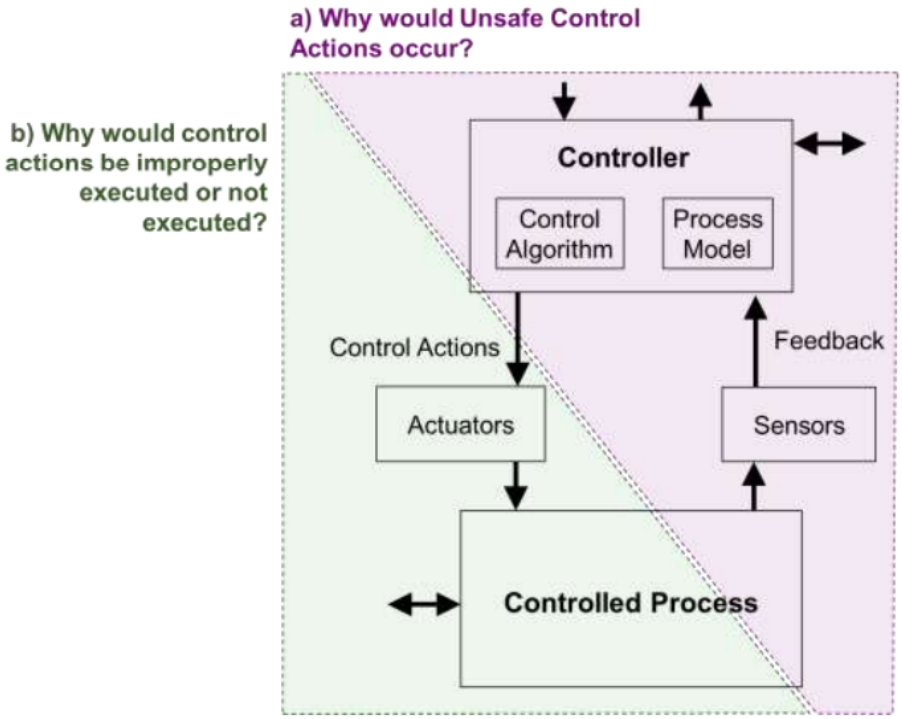

Figure 2.17: Two types of scenarios that must be considered

Figure A.4: Loss Scenario guidance diagram

Appendix B

Additional STAMP Ontology Reasoning

The decision to focus on STAMP was taken due to an existing set-theoretic representation from Thomas (2013). This section is included to show how the ontology terms could be used for set-theoretic reasoning. It demonstrates that the STAMP ontology is capable of reasoning to provide likely definitions of terms based on information already captured. Whilst such a feature is likely to be useful to the attentive professional, it apparently accomplishes some of the tasks the STPA learner is to learn how to do. Therefore these kinds of reasoning are not provided by default, being reserved for a high-level of interference when the user is in "trouble" in the Contingent Scaffolding framework.

B.1 Reasoning via Set Construction

Reasoning through constructing sets from the known properties of known individuals provides a means to identify the sets of interest to the analyst or supporting software. It may also be used to check if declared sets contain valid and all individuals. The utility of set-theoretic reasoning has already been established for STPA (Thomas 2013; Gurgel et al. 2015), here additional applications exploiting the defined ontology are presented.

Typically when considering ontology from a set-theoretic perspective, it's the individual instances of a class that are considered as the individuals within or without a particular set. When needing to consider those individuals as well as the classes themselves the computation becomes

prohibitively complex, entering the domain of second-order logic. However, for the STPA use-case there are no individuals to consider, only the classes themselves due to the hypothetical nature of the exercise. Therefore, for the set reasoning within this work, the classes are taken as the individuals, keeping within first-order logic. Thus a term like 'Hazard' is an individual belonging to the set of terms that are subsumed by 'Loss'.

B.2 Set Building Notation

Set-builder notation is used to describe these sets, which includes the typical set operators:

- $\wedge$ Union
- $\vee$ Disjunction
- $\setminus$ Subtraction

The building of a set contains a variable found on both sides of a dividing |, which can be read as "for". The left hand side can be just the variable or an expression denoting the set, the right hand side is another expression or predicate that determines the individuals that may be unified with the variable for inclusion. The syntax informally corresponds to:

$$\{\texttt{how the set is built}|\texttt{the individuals the set is built from}\}$$

For example, the building of a set of facts within a `sit/1` predicate, $sit(t)$, where the terms t have been defined as a subClassOf Situation would be written as so:

$$\{sit(t)|subClassOf(Situation, t)\}$$

B.3 Building Sets for STPA Support

Safety constraint situations can often be determined by negating some fluent that holds in a Hazard (N. Leveson and Thomas 2018), which is a trivial task for software. The following set constructor outlines the reasoning required to achieve this, the right-hand side enumerates the subclasses of Hazard and the Fluents that hold in each one. The left-hand side relates a set of Fluents with the Hazard by a `mayPrevent/2` predicate, which is used so that it may be suggested to the analyst that the set of Fluents might prevent the Hazard from occurring. The

set of Fluents within `mayPrevent/2` is the set that hold in the Hazard with the Fluent under consideration subtracted and its negation added. The result is a set of `mayPrevent/2` predicates that can be presented to the analyst for consideration and added to their ontology if approved, with the addition of an identity for the Safety Constraint Situation they represent.

$$\{\texttt{mayPrevent}(\{\neg f\} \vee \{f' | \texttt{holdsIn}(f', h)\} \setminus \{f\}, h) | \texttt{subClassOf}(h, \texttt{Hazard}) \wedge \texttt{holdsIn}(f, h)\}$$

Similar to the generation of possible Safety Constraint Situations, it's also possible to determine the set of fluents that do not yet have a control action defined by subtracting the set that do from the whole set of fluents. Although this can aid the analyst in ensuring control actions aren't missed, it's non-exhaustive. It only determines if every fluent can be controlled by at least one control action, rather than if all control actions have been defined for all fluents, which cannot be known by the software. Therefore it's best used to ensure each fluent has been considered. The same can be applied to the feedback side of the control loop with trivial adjustment.

$$\{f | \texttt{subClassOf}(f, \texttt{Fluent}) \wedge f \neq \texttt{Fluent}\} \setminus \{f | \texttt{subClassOf}(c, \texttt{ControlAction}) \wedge \texttt{requestsEffect}(c, f)\}$$

An intelligent context situation for unsafe control actions can be determined based on the explanation of why that control action is unsafe by unifying fluents that are required for the action to be possible with those that hold in the appropriate hazard or safety constraint situation minus the appropriate ones the control action effects.

For example, UCA-1 turning on the power source, is potentially hazardous as it is a control action that requests that the power be on, which holds in the H-1 hazard. For the case of providing a control action that causes a fluent to hold in the hazard, such as this one, the context situation can be generated with:

$$\{f | \texttt{isPossibleIn}(\texttt{TurnOnPower}, s) \wedge \texttt{holdsIn}(f, s)\} \vee (\{f | \texttt{holdsIn}(f, \texttt{H-1})\} \setminus \{f | \texttt{requestsPositiveEffect}(\texttt{TurnOnPower}, f)\})$$

The generated context to provide is more explicit than given in the standard STPA format as

it ensures the action is possible:

$$\{\texttt{PowerSourceOff}\} \vee (\{\texttt{DoorOpenFluent}, \texttt{PowerSourceOn}\} \setminus \{\texttt{PowerSourceOn}\})$$
$$= \{\texttt{PowerSourceOff}, \texttt{DoorOpenFluent}\}$$

Appendix C

Missing and Mistake Interventions in Prolog

```
missing(step1,
  'Hazards will lead to a loss in some worst-case environment',
  logged_assertion(Hazard, subClassOf, 'Hazard') and
  not (logged_assertion(Loss, subClassOf, 'Loss') and
  logged_assertion(Hazard, hasPossible, Loss)),
  'Have you not asserted which Loss is possible to occur in the
  ~s Hazard?'-[Hazard]
).

missing(step1,
  'Hazards must describe states or conditions to be prevented',
  logged_assertion(Hazard, subClassOf, 'Hazard') and
  not (logged_assertion(Hazard, hasHolding, _) or
  logged_assertion(Hazard, notHasHolding, _)),
  'Have you not asserted any fluents (conditions) that must hold
  in the ~s Hazard?'-[Hazard]
).

mistake('Hazards should not include ambiguous or recursive words
```

```
  like "unsafe", "unintended", "accidental", etc.',
  logged_assertion(Hazard, subClassOf, 'Hazard') and
  ( logged_assertion(Hazard, hasHolding, Fluent)
  or logged_assertion(Hazard, notHasHolding, Fluent)
  ) and
  logged_assertion(Fluent,
      hasQualityInSomeSituation, Quality) and
  logged_assertion(Quality, label, Label) and
  fact(Label, containsWord, Word) and
  fact(Word, instanceOf, 'AmbiguousOrRecursiveWord'),
  'Haven\'t you defined the Hazard ~s with "~s", which
  contains the ambiguous or recursive word "~s"?'-[
  Hazard, Label, Word]
).

mistake('Avoid using ambiguous and vague labels in the control
  structure: "commands", "feedback", "status", "computer"',
  logged_assertion(CA, subClassOf, 'ControlAction') and
  logged_assertion(CA, label, Label) and
  fact(Label, containsWord, Word) and
  fact(Word, instanceOf, 'AmbiguousOrRecursiveWord'),
  'Haven\'t you labelled the Control Action ~s "~s", which
  contains the ambiguous or recursive word "~s"?'-[
  CA, Label, Word]
).

mistake('Avoid using ambiguous and vague labels in the control
  structure: "commands", "feedback", "status", "computer"',
  logged_assertion(Feedback, subClassOf, 'Feedback') and
  logged_assertion(Feedback, label, Label) and
  fact(Label, containsWord, Word) and
  fact(Word, instanceOf, 'AmbiguousOrRecursiveWord'),
```

```
  'Haven\'t you labelled the Feedback ~s "~s", which contains
  the ambiguous or recursive word "~s"?'-[Feedback, Label, Word]
).

missing(step2,
  'Check that every controlled physical process is controlled
  by one or more controllers',
  logged_assertion(CP, subClassOf, 'ControlledProcess') and
  not (logged_assertion(CP, hasSubject, E) and
  logged_assertion(F, hasBearer, E) and
  fact(CA, requestsEffect, F) and
  logged_assertion(_, hasCapability, CA)),
  'Have you not asserted that any Controllers are capable of a
  Control Action that has an Effect upon some System that is
  subject to the Controlled Process ~s?'-[CP]
).

missing(step2,
  'Check that control actions needed to satisfy the
  responsibilities are included',
  logged_assertion(Hazard, subClassOf, 'Hazard') and
  logged_assertion(Hazard, hasHolding, Fluent) and
  not (logged_assertion(_, requestsToHold, Fluent)),
  'Your ~s Hazard has holding the ~s Fluent. Have you not
  asserted that any Control Action requests ~s to hold?'-[
  Hazard, Fluent, Fluent]
).

missing(step2,
  'Check that feedback needed to satisfy the responsibilities is
  included',
  logged_assertion(Hazard, subClassOf, 'Hazard') and
```

```
  logged_assertion(Hazard, hasHolding, Fluent) and
  not (logged_assertion(_, recordsFluent, Fluent)),
  'Your ~s Hazard has holding the ~s Fluent. Have you not
  asserted that any Feedback records the Fluent ~s?'-[
  Hazard, Fluent, Fluent]
).

missing(step3,
  'Ensure traceability is documented to link every unsafe control
  action with one or more hazards',
  logged_assertion(CA, subClassOf,
      'ProvidingPotentiallyHazardousControlAction') and
  not logged_assertion(CA, providingPotentiallyLeadsTo, _Hazard),
  'Have you not asserted that providing the unsafe control action
  "~s" can lead to some hazard?'-[CA]
).

missing(step3,
  'Ensure traceability is documented to link every unsafe control
  action with one or more hazards',
  logged_assertion(CA, subClassOf,
      'NotProvidingPotentiallyHazardousControlAction') and
  not logged_assertion(CA, notProvidingPotentiallyLeadsTo, _Hazard),
  'Have you not asserted that not providing the unsafe control action
  "~s" can lead to some hazard?'-[CA]
).

missing(step3,
  'Ensure every unsafe control action specifies the context that
  makes the control action unsafe',
  fact(CA, subClassOf, 'PotentiallyHazardousControlAction') and
  fact(CA, is_id, 'ControlAction') and
```

```
  not logged_assertion(CA, isPossibleIn, _),
  'You\'ve asserted that the ~s Control Action is potentially
  hazardous. Have you not asserted in what Situation it\'s possible
  for ~s to occur?'-[CA, CA]
).

missing(step3,
  'Ensure the unsafe control action contexts are defined clearly',
  fact(CA, subClassOf, 'PotentiallyHazardousControlAction') and
  fact(CA, is_id, 'ControlAction') and
  logged_assertion(CA, isPossibleIn, S) and
  not ( logged_assertion(S, hasHolding, _)
    or logged_assertion(S, notHasHolding, _)
  ),
  'You\'ve asserted that the ~s Control Action is potentially
  hazardous. Have you not described the Fluents that hold in
  the Situation ~s in which ~s is possible?'-[CA, S, CA]
).

missing(step1,
  'Step 1 precedes Step 2',
  not(logged_assertion(_L, subClassOf, 'Loss') and
  logged_assertion(_H, subClassOf, 'Hazard')),
  'Have you not asserted any Losses or Hazards yet?'-[]
).

missing(step2,
  'Step 2 precedes Step 3',
  not(
    logged_assertion(_E, hasCapability, _CA) and
    fact(_CA, requestsEffect, _Fluent) and
    logged_assertion(_FB, subClassOf, 'Feedback')
```

```
  ),
  'Have you not asserted any Controller can do some Control Action
  that request an effect or any Feedback yet?'-[]
).

missing(step3,
  'Identify all Providing Potentially Hazardous Control Actions',
  fact(CA, equivalentTo,
    'ProvidingPotentiallyHazardousControlAction') and
  not logged_assertion(CA, subClassOf,
        'ProvidingPotentiallyHazardousControlAction'),
  'Have you not asserted that ~s is a Providing Potentially
  Hazardous Control Action?'-[CA]
).

missing(step3,
  'Identify all Not Providing Potentially Hazardous Control
  Actions',
  fact(CA, equivalentTo,
    'NotProvidingPotentiallyHazardousControlAction') and
  not logged_assertion(CA, subClassOf,
    'NotProvidingPotentiallyHazardousControlAction'),
    'Have you not asserted that ~s is a Not Providing
    Potentially Hazardous Control Action?'-[CA]
).

mistake('If you have more than 7 hazards, consider grouping or
  combining them to create a more manageable set',
  fact('Hazard', subClassCount, C) and C > 7,
  'You\'ve asserted ~d Hazards, have you considered reducing the
  number you are considering in this analysis?'-[C]
).
```

```
mistake('Fluent can\'t both hold and not hold in a situation',
  logged_assertion(Sit, notHasHolding, Fluent) and
  logged_assertion(Sit, hasHolding, Fluent),
  'Have you asserted that the Fluent ~s both holds and doesn\'t
  hold in the Situation ~s?'-[Fluent, Sit]
).

mistake('Control Action can\'t both cause a fluent to hold and
  not hold',
  logged_assertion(CA, subClassOf, 'ControlAction') and
  logged_assertion(CA, requestsToHold, F) and
  logged_assertion(CA, requestsToNotHold, F),
  'Have you asserted that the Control Action ~s requests ~F both
  holds and doesn\'t hold?'-[CA, F]
).
```

Appendix D

User Study Protocol

This Appendix contains documents provided to participants.

D.1 Participant Information

You are being invited to take part in this research. Before you decide it is important for you to understand why the research is being done and what it will involve. Please take time to read the following information carefully and discuss it with others if you wish. Ask us if there is anything that is not clear or if you would like more information. Take time to decide whether or not you wish to take part.

Conducting a safety analysis is a difficult task, particularly for those who haven't developed expertise over years of practice. Therefore we've developed a software application to help those who are still learning. This software is based upon AI technologies like Ontology, and uses teaching support methods like Contingent Scaffolding. We hope that it will help a non-expert analyst to work more like an expert.

D.1.1 Taking Part

As a participant you'll be contributing to research that will help those who conduct safety analysis do a better job to save lives, money, the environment, etc. Plus you'll be reimbursed with a £25 Amazon voucher.

It is up to you to decide whether or not to take part. If you do decide to take part you will need to provide consent when you begin, and you can still withdraw at any time during the study,

just click the "Withdraw" link. You do not have to give a reason for withdrawal. However, once you have completed the study there will be no way to withdraw as all of the data is collected anonymously.

We want to know if the tutor part of the AI technology can provide useful support, so we need to compare using the software with and without the tutor. To do this we'll randomly turn the AI tutor on or off when you start the study. The tutor gives interventions, so if you don't see any that either means you're doing really well or it's been turned off. But even if you don't have the tutor to help you, we still want to know if the software has been useful to you.

D.1.2 What Do I Need To Do?

The study will require an hour of your time and can be completed entirely online with a laptop or desktop computer. It's not recommended to use a tablet for this due to the amount of typing involved.

The STPA analysis is to be conducted for the provided system and scenario, a description of which you should have received along with this information sheet, please contact us for a copy if you do not have it. This description will also be reiterated during the study to refresh your memory. A researcher will be available to answer system and study questions during University of Leeds office hours throughout the period when the study is open.

You will be expected to complete the study in a single sitting, it is not possible to guarantee a session can be resumed once stopped, which means you would have to start all over again. Don't worry though, a simple page refresh is fine as we'll be able to identify you via your session data.

D.1.3 What Will Happen To My Data?

We gather three sources of data.

If you choose to enter the prize-draw we'll ask you for your University of Leeds email address so we can distribute the prizes. This will be kept strictly confidential, stored separately from the research data. Your email address will be deleted as soon as all three prizes have been distributed.

The other two types of data we gather are your answers to the questionnaires and the log of actions taken during the STPA analysis. These are gathered in the software application with no

identifying information. There's no way we can tell who you are from this data.

The actions we log during the analysis tell us what facts you're asserting and retracting, what you're looking at on the screen, and any AI tutor intervention interactions. An example log looks like this:

```
focus_concept('Hazard', 10, datetime(2021, 2, 12, 10, 15, 58)).
```

In this log, user 10, whoever that may be, is looking at their Hazards. Should you wish to know more about these logs, please do not hesitate to contact us.

The gathered data will be analysed, with the results to be published at relevant conferences and journals, as well as within the final thesis. As we believe in open-research and open-data, we will also publish the anonymous data to Research Data Leeds so that other researchers can make use of it. This includes both the logs and answers to the questionnaires. Should you wish to be notified of this publication, whether participating in the study or not, please contact us.

Note, if you withdraw consent, all data associated with that your user number will be automatically deleted. Although identifying information is not required anywhere in this part of the process, should you include any in your answers to the questionnaires or in your STPA analysis it will be redacted prior to publication.

During the study and analysis, all data transmission will be conducted via encrypted channels, such as HTTPS. To prevent the accidental loss of your data we will back-up the server running the software. This back-up will be deleted upon publication of the data.

D.1.4 Who Is Doing This Research?

We are very grateful for the UKRC and DSTL for funding for this research, and to the DSTL for their additional consultation and initial impetus.

The primary researcher and initial point of contact is:

- Paul Brown, University of Leeds, PhD researcher, (sc16pb@leeds.ac.uk)

The supervision team are:

- Prof Vania Dimitrova, University of Leeds, (V.G.Dimitrova@leeds.ac.uk)
- Prof Anthony G. Cohn, University of Leeds
- Glen Hart, DSTL

D.1.5 Finally...

Thank you for reading! Please keep a copy of this for your records. You should also have received a copy of the consent form you'll be asked to complete on beginning the study, as well as a description of the system scenario for the analysis. Should you require any additional information, please contact Paul Brown (sc16pb@leeds.ac.uk).

D.2 Recommender System Scenario Safety Analysis

The University of Leeds is (hypothetically) considering developing a module recommender to aid students in choosing which modules to study. As part of their requirements gathering process they've asked you to conduct a System Safety Analysis, applying System-Theoretic Process Analysis (STPA), to ensure that their good intention works out.

This is a part of a wider analysis; you only need to consider this small, simple component.

D.2.1 System Safety Goals

This study will focus only on the primary goals of student and staff satisfaction. These goals will be considered failed if there is:

1. Decrease of student satisfaction
2. Decrease of staff satisfaction

To prevent failing the goals the following conditions of the module recommender must be met:

- an adequate user model, including interests and career goals
- adequate information regarding modules
- an appropriate recommendation algorithm

D.2.2 System Stakeholders

The project leader has also identified the following acting stakeholders, who are people with a vested interest in the success of the project:

- Student: one who chooses modules
- Module leader: one who describes the module
- Developer: one who authors the recommender software

D.2.3 The System

A broad system model is considered in this safety analysis. The system is the actors as well as the recommender software (see diagram below). At this early stage of design only a use-case diagram has been created showing the system interactions including actors and activities. This diagram contains sufficient information for the safety analysis: adding additional information is unnecessary for a successful safety analysis and may even lead to a poor safety analysis.

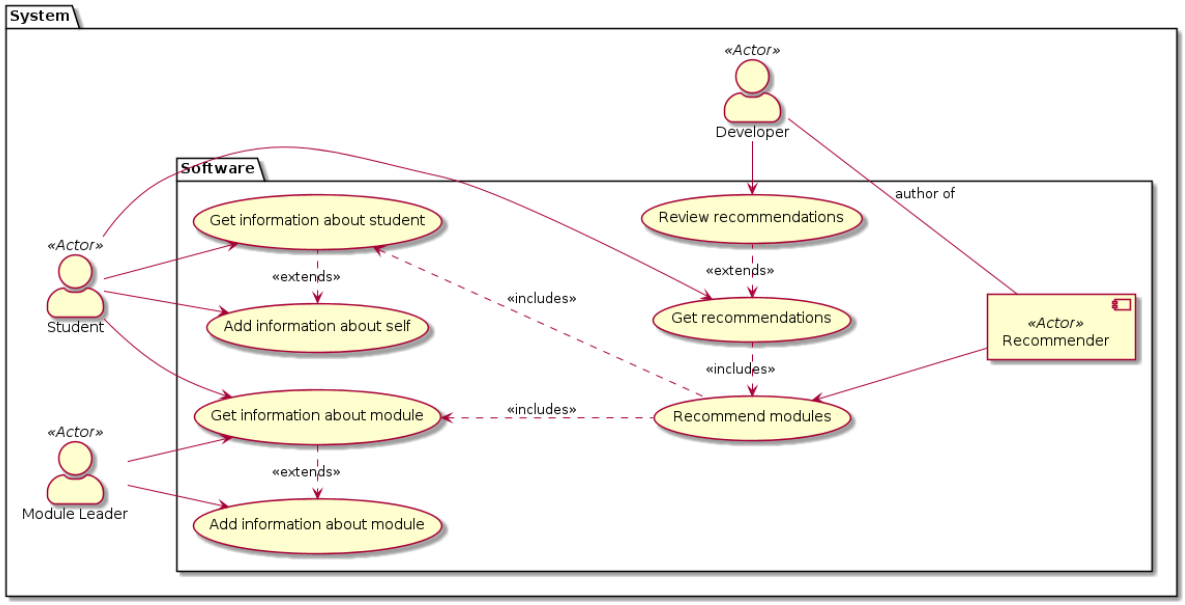

Figure D.1: Recommender System Use-Case Diagram (UML2)

D.2.4 Instructions

1. Go to https://owlsai.com
2. Follow the instructions there
3. Don't spend more than 2 hours on this, it should take about an hour
4. Keep this document handy and use it to help you with the analysis

D.3 STPA Questionnaire

These are the questions presented to a user before and after their analysis in order to assess learning. The correct answers are marked with a dot.

Do all definitions of hazards need to lead to a loss?

⊙ All hazard definitions must include a link to a loss

○ Not all hazard definitions need to include a link to a loss

Can we just use a hazard with a name or do we have to describe the hazardous situation?

○ A name is enough

⊙ Must decribe situation

Which hazard definition is better?

○ H-1: Power tool in unsafe state

⊙ H-2: Power tool safety guard not in place

The driver of a car can see how much fuel they have left in the tank. What is the best label for this feedback?

○ Fuel status

⊙ Quantity remaining

Is it necessary to define the context in which unsafe control actions are possible?

⊙ Yes

○ No

What do you need to define in step 1 of STPA?

⊡ Losses

☐ Controllers

☐ Loss Scenarios

☐ Controlled Processes

☐ Which Control Actions are Unsafe

☐ Control Actions

☐ Feedback

☑ Hazards

What do you need to define in step 2 of STPA?

☐ Losses

☑ Controllers

☐ Loss Scenarios

☑ Controlled Processes

☐ Which Control Actions are Unsafe

☑ Control Actions

☑ Feedback

☐ Hazards

What do you need to define in step 3 of STPA?

☐ Losses

☐ Controllers

☐ Loss Scenarios

☐ Controlled Processes

☑ Which Control Actions are Unsafe

☐ Control Actions

☐ Feedback

☐ Hazards

D.4 System Scenario Solution

This is derived from the system scenario. It's described here with a frame-based, turtle-like syntax to ease reading in textual format.

```
L–1 a Class
    subClassOf Loss
    label "Decrease of student satisfaction"

L–2 a Class
    subClassOf Loss
    label "Decrease of staff satisfaction"

H–1 a Class
    subClassOf Hazard
    hasHolding F–1
    hasPossible L–1
    hasPossible L–2

H–2 a Class
    subClassOf Hazard
    notHasHolding F–2
    hasPossible L–1
    hasPossible L–2

H–3 a Class
    subClassOf Hazard
    notHasHolding F–3
    hasPossible L–1
    hasPossible L–2

F–1 a Class
    subClassOf QualityFluent
```

```
    hasBearer E-1
    hasQualityInSomeSituation Q-1

F-2 a Class
    subClassOf QualityFluent
    hasBearer E-2
    hasQualityInSomeSituation Q-1

F-3 a Class
    subClassOf QualityFluent
    hasBearer E-3
    hasQualityInSomeSituation Q-2

E-1 a Class
    subClassOf Entity
    label "user model"

E-2 a Class
    subClassOf Entity
    label "module model"

E-3 a Class
    subClassOf Entity
    label "recommendation algorithm"

Q-1 a Class
    subClassOf Quality
    label "adequate"

Q-2 a Class
    subClassOf Quality
    label "appropriate"
```

```
CP–1 a Class
    subClassOf ControlledProcess
    label "Recommending Modules"
    hasSubject E–1
    hasSubject E–2
    hasSubject E–3

E–4 a Class
    subClassOf Entity
    label "Student"
    hasCapability CA–1
    hasCapability CA–2
    hasFeedback FB–1

E–5 a Class
    subClassOf Entity
    label "Module Leader"
    hasCapability CA–3
    hasCapability CA–4
    hasFeedback FB–2

E–6 a Class
    subClassOf Entity
    label "Developer"
    hasCapability CA–5
    hasCapability CA–6
    hasFeedback FB–1
    hasFeedback FB–2
    hasFeedback FB–3

CA–1 a Class
```

```
    subClassOf ControlAction
    subClassOf NotProvidingPotentiallyHazardousControlAction
    label "Add adequate information about self"
    requestsToHold F-1
    isPossibleIn S-1
    notProvidingPotentiallyLeadsTo H-1

CA-2 a Class
    subClassOf ControlAction
    subClassOf ProvidingPotentiallyHazardousControlAction
    label "Add inadequate information about self"
    requestsToNotHold F-1
    isPossibleIn S-1
    providingPotentiallyLeadsTo H-1

CA-3 a Class
    subClassOf ControlAction
    subClassOf NotProvidingPotentiallyHazardousControlAction
    label "Add adequate information about module"
    requestsToHold F-2
    isPossibleIn S-1
    notProvidingPotentiallyLeadsTo H-2

CA-4 a Class
    subClassOf ControlAction
    subClassOf ProvidingPotentiallyHazardousControlAction
    label "Add inadequate information about module"
    requestsToNotHold F-2
    isPossibleIn S-1
    providingPotentiallyLeadsTo H-2

CA-5 a Class
```

```
    subClassOf ControlAction
    subClassOf NotProvidingPotentiallyHazardousControlAction
    label "Author appropriate recommendation algorithm"
    requestsToHold F-3
    isPossibleIn S-1
    notProvidingPotentiallyLeadsTo H-3

CA-6 a Class
    subClassOf ControlAction
    subClassOf ProvidingPotentiallyHazardousControlAction
    label "Author inappropriate recommendation algorithm"
    requestsToNotHold F-3
    isPossibleIn S-1
    providingPotentiallyLeadsTo H-3

FB-1 a Class
    subClassOf Feedback
    label "Information contained in user model"
    recordsFluent F-1

FB-2 a Class
    subClassOf Feedback
    label "Information contained in module model"
    recordsFluent F-2

FB-3 a Class
    subClassOf Feedback
    label "Performance metrics for the model under testing"
    recordsFluent F-3

S-1 a Class
    subClassOf Situation
```

```
label "The top situation which subsumes all others"
```

Appendix E

User Study Selected Logs

E.1 User 43

```
assert(spo('L-1',type,'Class'),43,datetime(2022,3,9,12,50,55)).
assert(spo('L-1',subClassOf,'Loss'),43,datetime(2022,3,9,12,50,55)).
assert(spo('L-1',label,"Decrease"),43,datetime(2022,3,9,12,51,27)).
glossary_lookup('Loss',43,datetime(2022,3,9,12,51,27)).
retract(spo('L-1',label,"Decrease"),43,datetime(2022,3,9,12,52,0)).
assert(spo('L-1',label,"Decrease in student satisfaction"),
    43,datetime(2022,3,9,12,52,0)).
assert(spo('L-2',type,'Class'),43,datetime(2022,3,9,12,52,0)).
assert(spo('L-2',subClassOf,'Loss'),43,datetime(2022,3,9,12,52,0)).
assert(spo('L-2',label,"Decrease in "),43,datetime(2022,3,9,12,52,4)).
retract(spo('L-2',label,"Decrease in "),
    43,datetime(2022,3,9,12,52,17)).
assert(spo('L-2',label,"Decrease in staff satisfaction"),
    43,datetime(2022,3,9,12,52,17)).
assert(spo('H-1',type,'Class'),43,datetime(2022,3,9,12,52,17)).
assert(spo('H-1',subClassOf,'Hazard'),43,datetime(2022,3,9,12,52,17)).
assert(spo('F-1',type,'Class'),43,datetime(2022,3,9,12,52,41)).
assert(spo('F-1',subClassOf,'Fluent'),43,datetime(2022,3,9,12,52,41)).
assert(spo('E-1',type,'Class'),43,datetime(2022,3,9,12,52,41)).
```

```
assert(spo('E-1',subClassOf,'Entity'),43,datetime(2022,3,9,12,52,41)).
assert(spo('Q-1',type,'Class'),43,datetime(2022,3,9,12,52,41)).
assert(spo('Q-1',subClassOf,'Quality'),
    43,datetime(2022,3,9,12,52,41)).
assert(spo('H-1',hasHolding,'F-1'),43,datetime(2022,3,9,12,52,41)).
assert(spo('F-1',hasBearer,'E-1'),43,datetime(2022,3,9,12,52,41)).
assert(spo('F-1',hasQualityInSomeSituation,'Q-1'),
    43,datetime(2022,3,9,12,52,41)).
assert(spo('E-1',label,"Module information"),
    43,datetime(2022,3,9,12,54,27)).
assert(spo('Q-1',label,"unclear"),43,datetime(2022,3,9,12,54,35)).
assert(spo('H-1',hasPossible,'L-1'),43,datetime(2022,3,9,12,54,35)).
assert(spo('F-2',type,'Class'),43,datetime(2022,3,9,12,54,39)).
assert(spo('F-2',subClassOf,'Fluent'),43,datetime(2022,3,9,12,54,39)).
assert(spo('E-2',type,'Class'),43,datetime(2022,3,9,12,54,39)).
assert(spo('E-2',subClassOf,'Entity'),43,datetime(2022,3,9,12,54,39)).
assert(spo('Q-2',type,'Class'),43,datetime(2022,3,9,12,54,39)).
assert(spo('Q-2',subClassOf,'Quality'),43,datetime(2022,3,9,12,54,39)).
assert(spo('H-1',hasHolding,'F-2'),43,datetime(2022,3,9,12,54,39)).
assert(spo('F-2',hasQualityInSomeSituation,'Q-2'),
    43,datetime(2022,3,9,12,54,39)).
assert(spo('F-2',hasBearer,'E-2'),43,datetime(2022,3,9,12,54,39)).
retract(spo('F-2',hasQualityInSomeSituation,'Q-2'),
    43,datetime(2022,3,9,12,54,42)).
retract(spo('Q-2',subClassOf,'Quality'),
    43,datetime(2022,3,9,12,54,42)).
retract(spo('Q-2',type,'Class'),43,datetime(2022,3,9,12,54,42)).
retract(spo('E-2',subClassOf,'Entity'),
    43,datetime(2022,3,9,12,54,42)).
retract(spo('E-2',type,'Class'),43,datetime(2022,3,9,12,54,42)).
retract(spo('F-2',hasBearer,'E-2'),43,datetime(2022,3,9,12,54,42)).
retract(spo('F-2',subClassOf,'Fluent'),
```

```
    43,datetime(2022,3,9,12,54,42)).
retract(spo('F-2',type,'Class'),43,datetime(2022,3,9,12,54,42)).
retract(spo('H-1',hasHolding,'F-2'),43,datetime(2022,3,9,12,54,42)).
assert(spo('H-2',type,'Class'),43,datetime(2022,3,9,12,54,45)).
assert(spo('H-2',subClassOf,'Hazard'),43,datetime(2022,3,9,12,54,45)).
assert(spo('F-2',type,'Class'),43,datetime(2022,3,9,12,55,0)).
assert(spo('F-2',subClassOf,'Fluent'),43,datetime(2022,3,9,12,55,0)).
assert(spo('E-2',type,'Class'),43,datetime(2022,3,9,12,55,0)).
assert(spo('E-2',subClassOf,'Entity'),43,datetime(2022,3,9,12,55,0)).
assert(spo('Q-2',type,'Class'),43,datetime(2022,3,9,12,55,0)).
assert(spo('Q-2',subClassOf,'Quality'),43,datetime(2022,3,9,12,55,0)).
assert(spo('F-2',hasBearer,'E-2'),43,datetime(2022,3,9,12,55,0)).
assert(spo('F-2',hasQualityInSomeSituation,'Q-2'),
    43,datetime(2022,3,9,12,55,0)).
assert(spo('H-2',hasHolding,'F-2'),43,datetime(2022,3,9,12,55,0)).
assert(spo('E-2',label,"Recommendation algorithm"),
    43,datetime(2022,3,9,12,55,22)).
assert(spo('Q-2',label,"flawed"),43,datetime(2022,3,9,12,55,43)).
assert(spo('H-2',hasPossible,'L-1'),43,datetime(2022,3,9,12,55,43)).
assert(spo('H-2',hasPossible,'L-2'),43,datetime(2022,3,9,12,55,46)).
request_intervention(43,datetime(2022,3,9,12,56,41)).
glossary_lookup('Hazard',43,datetime(2022,3,9,12,56,46)).
focus_concept('ControlledProcess',43,datetime(2022,3,9,12,57,13)).
focus_step(step2,43,datetime(2022,3,9,12,57,13)).
focus_concept('Entity',43,datetime(2022,3,9,12,57,58)).
focus_concept('ControlledProcess',43,datetime(2022,3,9,12,58,2)).
assert(spo('CP-1',type,'Class'),43,datetime(2022,3,9,12,58,5)).
assert(spo('CP-1',subClassOf,'ControlledProcess'),
    43,datetime(2022,3,9,12,58,5)).
focus_subconcept('CP-1',43,datetime(2022,3,9,12,58,13)).
focus_concept('Entity',43,datetime(2022,3,9,12,58,31)).
focus_concept('ControlledProcess',43,datetime(2022,3,9,12,58,34)).
```

```
glossary_lookup('ControlledProcess',43,datetime(2022,3,9,12,58,40)).
focus_concept('Entity',43,datetime(2022,3,9,12,58,47)).
focus_concept('ControlledProcess',43,datetime(2022,3,9,12,58,50)).
retract(spo('CP-1',subClassOf,'ControlledProcess'),
    43,datetime(2022,3,9,12,58,51)).
retract(spo('CP-1',type,'Class'),43,datetime(2022,3,9,12,58,51)).
focus_concept('Entity',43,datetime(2022,3,9,12,58,52)).
focus_subconcept('E-1',43,datetime(2022,3,9,12,58,53)).
glossary_lookup('Entity',43,datetime(2022,3,9,12,58,56)).
focus_concept('ControlAction',43,datetime(2022,3,9,12,59,7)).
glossary_lookup('ControlAction',43,datetime(2022,3,9,12,59,10)).
assert(spo('CA-1',type,'Class'),43,datetime(2022,3,9,12,59,18)).
assert(spo('CA-1',subClassOf,'ControlAction'),
    43,datetime(2022,3,9,12,59,18)).
focus_subconcept('CA-1',43,datetime(2022,3,9,12,59,24)).
assert(spo('F-3',type,'Class'),43,datetime(2022,3,9,13,0,6)).
assert(spo('F-3',subClassOf,'Fluent'),43,datetime(2022,3,9,13,0,6)).
assert(spo('CA-1',requestsToHold,'F-3'),43,datetime(2022,3,9,13,0,6)).
assert(spo('F-3',hasBearer,'E-1'),43,datetime(2022,3,9,13,0,6)).
assert(spo('F-3',hasQualityInSomeSituation,'Q-2'),
    43,datetime(2022,3,9,13,0,6)).
assert(spo('CA-1',label,"Module Leader "),
    43,datetime(2022,3,9,13,0,37)).
retract(spo('CA-1',label,"Module Leader "),
    43,datetime(2022,3,9,13,0,48)).
assert(spo('CA-1',label,"Module Leader describes module"),
    43,datetime(2022,3,9,13,0,48)).
focus_concept('Entity',43,datetime(2022,3,9,13,0,59)).
focus_concept('ControlledProcess',43,datetime(2022,3,9,13,1,7)).
assert(spo('CP-1',type,'Class'),43,datetime(2022,3,9,13,1,10)).
assert(spo('CP-1',subClassOf,'ControlledProcess'),
    43,datetime(2022,3,9,13,1,10)).
```

```
focus_subconcept('CP-1',43,datetime(2022,3,9,13,1,14)).
glossary_lookup('ControlledProcess',43,datetime(2022,3,9,13,1,21)).
glossary_lookup('ControlledProcess',43,datetime(2022,3,9,13,1,22)).
assert(spo('CP-1',label,"Students choosing a module"),
    43,datetime(2022,3,9,13,2,4)).
assert(spo('CP-2',type,'Class'),43,datetime(2022,3,9,13,2,27)).
assert(spo('CP-2',subClassOf,'ControlledProcess'),
    43,datetime(2022,3,9,13,2,27)).
focus_subconcept('CP-2',43,datetime(2022,3,9,13,2,31)).
retract(spo('CP-2',subClassOf,'ControlledProcess'),
    43,datetime(2022,3,9,13,2,32)).
retract(spo('CP-2',type,'Class'),43,datetime(2022,3,9,13,2,32)).
focus_concept('Entity',43,datetime(2022,3,9,13,2,33)).
request_intervention(43,datetime(2022,3,9,13,2,51)).
intervene('Check that control actions needed to satisfy the \
    responsibilities are included',
    logged_assertion('H-2',subClassOf,'Hazard')
        and logged_assertion('H-2',hasHolding,'F-2')
        and not logged_assertion(_20334,requestsToHold,'F-2'),
    1,datetime(2022,3,9,13,2,51)).
focus_step(step1,43,datetime(2022,3,9,13,3,6)).
focus_concept('Loss',43,datetime(2022,3,9,13,3,6)).
focus_step(step2,43,datetime(2022,3,9,13,3,11)).
focus_concept('ControlledProcess',43,datetime(2022,3,9,13,3,11)).
focus_concept('ControlAction',43,datetime(2022,3,9,13,3,26)).
focus_concept('Entity',43,datetime(2022,3,9,13,3,28)).
assert(spo('E-1',hasCapability,'CA-1'),43,datetime(2022,3,9,13,3,37)).
retract(spo('E-1',hasCapability,'CA-1'),43,datetime(2022,3,9,13,4,1)).
assert(spo('E-3',type,'Class'),43,datetime(2022,3,9,13,4,8)).
assert(spo('E-3',subClassOf,'Entity'),43,datetime(2022,3,9,13,4,8)).
focus_subconcept('E-3',43,datetime(2022,3,9,13,4,10)).
assert(spo('E-3',label,"Module Leader"),
```

```
    43,datetime(2022,3,9,13,4,15)).
assert(spo('E-3',hasCapability,'CA-1'),43,datetime(2022,3,9,13,4,22)).
focus_concept('ControlAction',43,datetime(2022,3,9,13,4,24)).
focus_subconcept('CA-1',43,datetime(2022,3,9,13,4,25)).
retract(spo('CA-1',label,"Module Leader describes module"),
    43,datetime(2022,3,9,13,4,49)).
assert(spo('CA-1',label,"Describes module"),
    43,datetime(2022,3,9,13,4,49)).
focus_concept('Entity',43,datetime(2022,3,9,13,4,49)).
focus_concept('ControlledProcess',43,datetime(2022,3,9,13,5,48)).
focus_subconcept('CP-1',43,datetime(2022,3,9,13,5,57)).
focus_concept('Entity',43,datetime(2022,3,9,13,6,39)).
focus_concept('ControlledProcess',43,datetime(2022,3,9,13,6,42)).
focus_concept('ControlAction',43,datetime(2022,3,9,13,6,45)).
assert(spo('CA-2',type,'Class'),43,datetime(2022,3,9,13,6,55)).
assert(spo('CA-2',subClassOf,'ControlAction'),
    43,datetime(2022,3,9,13,6,55)).
focus_subconcept('CA-2',43,datetime(2022,3,9,13,6,57)).
assert(spo('CA-2',label,"Provides information"),
    43,datetime(2022,3,9,13,7,4)).
focus_subconcept('CA-1',43,datetime(2022,3,9,13,7,4)).
assert(spo('F-4',type,'Class'),43,datetime(2022,3,9,13,7,23)).
assert(spo('F-4',subClassOf,'Fluent'),43,datetime(2022,3,9,13,7,23)).
assert(spo('F-4',hasBearer,'E-2'),43,datetime(2022,3,9,13,7,23)).
assert(spo('F-4',hasQualityInSomeSituation,'Q-1'),
    43,datetime(2022,3,9,13,7,23)).
assert(spo('CA-2',requestsToHold,'F-4'),
    43,datetime(2022,3,9,13,7,23)).
focus_concept('Entity',43,datetime(2022,3,9,13,8,24)).
assert(spo('E-4',type,'Class'),43,datetime(2022,3,9,13,8,30)).
assert(spo('E-4',subClassOf,'Entity'),43,datetime(2022,3,9,13,8,30)).
focus_subconcept('E-4',43,datetime(2022,3,9,13,8,32)).
```

```
assert(spo('E-4',label,"Student"),43,datetime(2022,3,9,13,8,38)).
assert(spo('E-4',hasCapability,'CA-2'),43,datetime(2022,3,9,13,8,44)).
focus_concept('Feedback',43,datetime(2022,3,9,13,8,50)).
focus_concept('ControlledProcess',43,datetime(2022,3,9,13,8,52)).
focus_subconcept('CP-1',43,datetime(2022,3,9,13,8,57)).
request_intervention(43,datetime(2022,3,9,13,9,10)).
request_intervention(43,datetime(2022,3,9,13,9,12)).
request_help('Check that control actions needed to satisfy the \
    responsibilities are included',43,datetime(2022,3,9,13,9,23)).
focus_concept('ControlAction',43,datetime(2022,3,9,13,9,31)).
focus_step(step1,43,datetime(2022,3,9,13,9,38)).
focus_concept('Loss',43,datetime(2022,3,9,13,9,38)).
focus_step(step2,43,datetime(2022,3,9,13,9,47)).
focus_concept('ControlledProcess',43,datetime(2022,3,9,13,9,47)).
focus_subconcept('CA-2',43,datetime(2022,3,9,13,9,51)).
retract(spo('CA-2',requestsToHold,'F-
    4'),43,datetime(2022,3,9,13,10,3)).
retract(spo('F-4',hasBearer,'E-2'),43,datetime(2022,3,9,13,10,3)).
retract(spo('F-4',hasQualityInSomeSituation,'Q-1'),
    43,datetime(2022,3,9,13,10,3)).
retract(spo('F-4',subClassOf,'Fluent'),43,datetime(2022,3,9,13,10,3)).
retract(spo('F-4',type,'Class'),43,datetime(2022,3,9,13,10,3)).
assert(spo('F-4',type,'Class'),43,datetime(2022,3,9,13,10,14)).
assert(spo('F-4',subClassOf,'Fluent'),43,datetime(2022,3,9,13,10,14)).
assert(spo('CA-2',requestsToHold,'F-4'),
    43,datetime(2022,3,9,13,10,14)).
assert(spo('F-4',hasBearer,'E-2'),43,datetime(2022,3,9,13,10,14)).
assert(spo('F-4',hasQualityInSomeSituation,'Q-1'),
    43,datetime(2022,3,9,13,10,14)).
focus_concept('ControlledProcess',43,datetime(2022,3,9,13,10,18)).
focus_concept('Loss',43,datetime(2022,3,9,13,10,20)).
focus_step(step1,43,datetime(2022,3,9,13,10,20)).
```

```
retract(spo('Q-2',label,"flawed"),43,datetime(2022,3,9,13,10,26)).
assert(spo('Q-2',label,"unclear"),43,datetime(2022,3,9,13,10,26)).
focus_concept('ControlledProcess',43,datetime(2022,3,9,13,10,28)).
focus_step(step2,43,datetime(2022,3,9,13,10,28)).
focus_concept('Entity',43,datetime(2022,3,9,13,10,30)).
focus_concept('ControlAction',43,datetime(2022,3,9,13,10,32)).
focus_step(step1,43,datetime(2022,3,9,13,10,34)).
focus_concept('Loss',43,datetime(2022,3,9,13,10,34)).
focus_concept('ControlledProcess',43,datetime(2022,3,9,13,10,46)).
focus_step(step2,43,datetime(2022,3,9,13,10,46)).
focus_concept('Feedback',43,datetime(2022,3,9,13,11,4)).
focus_concept('ControlAction',43,datetime(2022,3,9,13,11,8)).
retract(spo('CA-2',requestsToHold,'F-4'),
    43,datetime(2022,3,9,13,11,17)).
retract(spo('F-4',hasBearer,'E-2'),43,datetime(2022,3,9,13,11,17)).
retract(spo('F-4',hasQualityInSomeSituation,'Q-1'),
    43,datetime(2022,3,9,13,11,17)).
retract(spo('F-4',subClassOf,'Fluent'),
    43,datetime(2022,3,9,13,11,17)).
retract(spo('F-4',type,'Class'),43,datetime(2022,3,9,13,11,17)).
focus_step(step1,43,datetime(2022,3,9,13,11,18)).
focus_concept('Loss',43,datetime(2022,3,9,13,11,18)).
focus_concept('ControlledProcess',43,datetime(2022,3,9,13,11,22)).
focus_step(step2,43,datetime(2022,3,9,13,11,22)).
focus_subconcept('CA-2',43,datetime(2022,3,9,13,11,31)).
assert(spo('CA-2',requestsToHold,'F-2'),
    43,datetime(2022,3,9,13,11,36)).
request_intervention(43,datetime(2022,3,9,13,11,42)).
intervene('Check that control actions needed to satisfy the \
    responsibilities are included',
    logged_assertion('H-1',subClassOf,'Hazard')
        and logged_assertion('H-1',hasHolding,'F-1')
```

```
        and not logged_assertion(_10202,requestsToHold,'F-1'),
    1,datetime(2022,3,9,13,11,42)).
request_help('Check that control actions needed to satisfy the \
    responsibilities are included',43,datetime(2022,3,9,13,11,47)).
focus_subconcept('CA-1',43,datetime(2022,3,9,13,11,54)).
retract(spo('CA-1',requestsToHold,'F-3'),
    43,datetime(2022,3,9,13,11,59)).
retract(spo('F-3',hasBearer,'E-1'),43,datetime(2022,3,9,13,11,59)).
retract(spo('F-3',hasQualityInSomeSituation,'Q-2'),
    43,datetime(2022,3,9,13,11,59)).
retract(spo('F-3',subClassOf,'Fluent'),
    43,datetime(2022,3,9,13,11,59)).
retract(spo('F-3',type,'Class'),43,datetime(2022,3,9,13,11,59)).
assert(spo('CA-1',requestsToHold,'F-1'),
    43,datetime(2022,3,9,13,12,4)).
request_intervention(43,datetime(2022,3,9,13,12,8)).
intervene('Check that feedback needed to satisfy the \
    responsibilities is included',
    logged_assertion('H-1',subClassOf,'Hazard')
        and logged_assertion('H-1',hasHolding,'F-1')
        and not logged_assertion(_10570,recordsFluent,'F-1'),
    1,datetime(2022,3,9,13,12,8)).
request_help('Check that feedback needed to satisfy the \
    responsibilities is included',43,datetime(2022,3,9,13,12,11)).
focus_concept('Feedback',43,datetime(2022,3,9,13,12,16)).
glossary_lookup('Feedback',43,datetime(2022,3,9,13,12,18)).
assert(spo('FB-1',type,'Class'),43,datetime(2022,3,9,13,12,21)).
assert(spo('FB-1',subClassOf,'Feedback'),
    43,datetime(2022,3,9,13,12,21)).
focus_subconcept('FB-1',43,datetime(2022,3,9,13,12,23)).
focus_concept('Entity',43,datetime(2022,3,9,13,13,2)).
focus_concept('Feedback',43,datetime(2022,3,9,13,13,12)).
```

```
assert(spo('FB-1',label,"Read module information"),
    43,datetime(2022,3,9,13,13,56)).
assert(spo('FB-1',recordsFluent,'F-1'),
    43,datetime(2022,3,9,13,13,56)).
focus_concept('ControlledProcess',43,datetime(2022,3,9,13,14,10)).
focus_concept('Feedback',43,datetime(2022,3,9,13,14,12)).
focus_concept('ControlAction',43,datetime(2022,3,9,13,14,21)).
focus_concept('Entity',43,datetime(2022,3,9,13,14,31)).
assert(spo('E-4',informedBy,'FB-1'),43,datetime(2022,3,9,13,15,2)).
focus_concept('ControlledProcess',43,datetime(2022,3,9,13,15,37)).
focus_subconcept('CP-1',43,datetime(2022,3,9,13,15,40)).
glossary_lookup('ControlledProcess',43,datetime(2022,3,9,13,15,49)).
request_intervention(43,datetime(2022,3,9,13,15,54)).
intervene('Check that feedback needed to satisfy the \
    responsibilities is included',
    logged_assertion('H-2',subClassOf,'Hazard')
        and logged_assertion('H-2',hasHolding,'F-2')
        and not logged_assertion(_11228,recordsFluent,'F-2'),
    1,datetime(2022,3,9,13,15,54)).
request_help('Check that feedback needed to satisfy the \
    responsibilities is included',43,datetime(2022,3,9,13,15,56)).
focus_concept('Feedback',43,datetime(2022,3,9,13,16,0)).
assert(spo('FB-2',type,'Class'),43,datetime(2022,3,9,13,16,5)).
assert(spo('FB-2',subClassOf,'Feedback'),
    43,datetime(2022,3,9,13,16,5)).
focus_step(step1,43,datetime(2022,3,9,13,16,8)).
focus_concept('Loss',43,datetime(2022,3,9,13,16,8)).
focus_concept('ControlledProcess',43,datetime(2022,3,9,13,16,14)).
focus_step(step2,43,datetime(2022,3,9,13,16,14)).
focus_subconcept('FB-2',43,datetime(2022,3,9,13,16,18)).
assert(spo('FB-2',label,"Module suggested"),
    43,datetime(2022,3,9,13,16,46)).
```

```
assert(spo('FB-2',recordsFluent,'F-2'),
    43,datetime(2022,3,9,13,16,47)).
focus_concept('Entity',43,datetime(2022,3,9,13,16,50)).
focus_subconcept('E-4',43,datetime(2022,3,9,13,16,53)).
assert(spo('E-4',informedBy,'FB-2'),43,datetime(2022,3,9,13,17,5)).
request_intervention(43,datetime(2022,3,9,13,17,19)).
intervene('Check that every controlled physical process is \
    controlled by one or more controllers',
    logged_assertion('CP-1',subClassOf,'ControlledProcess')
        and not ( logged_assertion('CP-1',hasSubject,_11750)
            and logged_assertion(_11802,hasBearer,_11750)
            and fact(_11858,requestsEffect,_11802)
            and logged_assertion(_11894,hasCapability,_11858)
            ),
    1,datetime(2022,3,9,13,17,19)).
request_help('Check that every controlled physical process is \
    controlled by one or more controllers',
        43,datetime(2022,3,9,13,17,25)).
focus_concept('ControlledProcess',43,datetime(2022,3,9,13,17,39)).
glossary_lookup('ControlledProcess',43,datetime(2022,3,9,13,17,41)).
focus_concept('ControlAction',43,datetime(2022,3,9,13,17,57)).
focus_concept('ControlledProcess',43,datetime(2022,3,9,13,18,5)).
assert(spo('CP-1',hasSubject,'E-4'),43,datetime(2022,3,9,13,18,14)).
retract(spo('CP-1',hasSubject,'E-4'),43,datetime(2022,3,9,13,18,53)).
assert(spo('CP-1',hasSubject,'E-3'),43,datetime(2022,3,9,13,18,53)).
retract(spo('CP-1',hasSubject,'E-3'),43,datetime(2022,3,9,13,19,4)).
assert(spo('CP-1',hasSubject,'E-2'),43,datetime(2022,3,9,13,19,4)).
request_intervention(43,datetime(2022,3,9,13,19,28)).
focus_step(step3,43,datetime(2022,3,9,13,19,30)).
glossary_lookup('UCA',43,datetime(2022,3,9,13,19,36)).
assert(spo('CA-1',subClassOf,
        'ProvidingPotentiallyHazardousControlAction'),
```

```
                43,datetime(2022,3,9,13,19,44)).
assert(spo('CA-1',providingPotentiallyLeadsTo,'H-1'),
    43,datetime(2022,3,9,13,19,49)).
assert(spo('CA-1',subClassOf,
           'NotProvidingPotentiallyHazardousControlAction'),
                43,datetime(2022,3,9,13,19,59)).
assert(spo('CA-1',notProvidingPotentiallyLeadsTo,'H-1'),
    43,datetime(2022,3,9,13,20,3)).
assert(spo('CA-2',subClassOf,
           'NotProvidingPotentiallyHazardousControlAction'),
                43,datetime(2022,3,9,13,20,13)).
assert(spo('CA-2',notProvidingPotentiallyLeadsTo,'H-2'),
    43,datetime(2022,3,9,13,20,23)).
assert(spo('CA-2',subClassOf,
           'ProvidingPotentiallyHazardousControlAction'),
                43,datetime(2022,3,9,13,20,25)).
assert(spo('CA-2',providingPotentiallyLeadsTo,'H-2'),
    43,datetime(2022,3,9,13,20,27)).
assert(spo('CA-1',notProvidingPotentiallyLeadsTo,'H-2'),
    43,datetime(2022,3,9,13,20,48)).
assert(spo('CA-1',providingPotentiallyLeadsTo,'H-2'),
    43,datetime(2022,3,9,13,21,2)).
request_intervention(43,datetime(2022,3,9,13,21,7)).
intervene('Ensure every unsafe control action specifies the context \
    that makes the control action unsafe',
    fact('CA-2',subClassOf,'PotentiallyHazardousControlAction')
        and fact('CA-2',is_id,'ControlAction')
        and not logged_assertion('CA-2',isPossibleIn,_13440),
    1,datetime(2022,3,9,13,21,7)).
focus_step(step2,43,datetime(2022,3,9,13,21,43)).
focus_concept('ControlledProcess',43,datetime(2022,3,9,13,21,43)).
focus_concept('ControlAction',43,datetime(2022,3,9,13,21,44)).
```

```
focus_subconcept('CA-1',43,datetime(2022,3,9,13,22,0)).
focus_subconcept('CA-2',43,datetime(2022,3,9,13,22,54)).
focus_step(step3,43,datetime(2022,3,9,13,23,16)).
request_help('Ensure every unsafe control action specifies the \
    context that makes the control action unsafe',
        43,datetime(2022,3,9,13,23,20)).
request_help('Ensure every unsafe control action specifies the \
    context that makes the control action unsafe',
        43,datetime(2022,3,9,13,23,25)).
focus_step(step2,oswin,datetime(2022,3,9,13,23,27)).
focus_concept('Control Actions',oswin,datetime(2022,3,9,13,23,27)).
focus_subconcept('CA-2',oswin,datetime(2022,3,9,13,23,27)).
request_help('Ensure every unsafe control action specifies the \
    context that makes the control action unsafe',
        43,datetime(2022,3,9,13,23,27)).
assert(spo('S-1',type,'Class'),43,datetime(2022,3,9,13,23,44)).
assert(spo('S-1',subClassOf,'Situation'),
    43,datetime(2022,3,9,13,23,44)).
assert(spo('CA-2',isPossibleIn,'S-1'),43,datetime(2022,3,9,13,23,44)).
focus_subconcept('S-1',43,datetime(2022,3,9,13,23,54)).
focus_subconcept('CA-1',43,datetime(2022,3,9,13,23,59)).
assert(spo('S-2',type,'Class'),43,datetime(2022,3,9,13,24,11)).
assert(spo('S-2',subClassOf,'Situation'),
    43,datetime(2022,3,9,13,24,11)).
assert(spo('CA-1',isPossibleIn,'S-2'),43,datetime(2022,3,9,13,24,11)).
focus_step(step3,43,datetime(2022,3,9,13,24,15)).
request_intervention(43,datetime(2022,3,9,13,24,17)).
intervene('Ensure the unsafe control action contexts are defined \
    clearly',
    fact('CA-2',subClassOf,'PotentiallyHazardousControlAction')
        and fact('CA-2',is_id,'ControlAction')
        and logged_assertion('CA-2',isPossibleIn,'S-1')
```

```
        and not ( logged_assertion('S-1',hasHolding,_14256)
                or logged_assertion('S-1',notHasHolding,_14292)
                ),
    1,datetime(2022,3,9,13,24,17)).
focus_step(step4,43,datetime(2022,3,9,13,24,33)).
focus_step(step4,43,datetime(2022,3,9,13,24,37)).
intervention_feedback(intervention_feedback('Ensure the unsafe \
    control action contexts are defined clearly','Inappopriate'),
        43,datetime(2022,3,9,13,24,58)).
dismiss_intervention('Ensure the unsafe control action contexts are \
    defined clearly',43,datetime(2022,3,9,13,24,58)).
focus_step(step4,43,datetime(2022,3,9,13,25,1)).
focus_step(step3,43,datetime(2022,3,9,13,25,3)).
focus_step(step4,43,datetime(2022,3,9,13,25,4)).
intervene('Ensure the unsafe control action contexts are defined \
    clearly',
    fact('CA-1',subClassOf,'PotentiallyHazardousControlAction')
        and fact('CA-1',is_id,'ControlAction')
        and logged_assertion('CA-1',isPossibleIn,'S-2')
        and not ( logged_assertion('S-2',hasHolding,_13520)
                or logged_assertion('S-2',notHasHolding,_13528)
                ),
    1,datetime(2022,3,9,13,25,1)).
focus_step(step4,43,datetime(2022,3,9,13,25,7)).
focus_step(step3,43,datetime(2022,3,9,13,25,17)).
```

E.2 User 48

```
glossary_lookup('Loss',48,datetime(2022,3,10,9,50,9)).
assert(spo('L-1',type,'Class'),48,datetime(2022,3,10,9,50,27)).
assert(spo('L-1',subClassOf,'Loss'),48,datetime(2022,3,10,9,50,27)).
assert(spo('L-2',type,'Class'),48,datetime(2022,3,10,9,51,6)).
```

```
assert(spo('L-2',subClassOf,'Loss'),48,datetime(2022,3,10,9,51,6)).
retract(spo('L-2',subClassOf,'Loss'),48,datetime(2022,3,10,9,51,9)).
retract(spo('L-2',type,'Class'),48,datetime(2022,3,10,9,51,9)).
assert(spo('L-1',label,"Student satisfaction"),
    48,datetime(2022,3,10,9,51,29)).
assert(spo('L-2',type,'Class'),48,datetime(2022,3,10,9,51,29)).
assert(spo('L-2',subClassOf,'Loss'),48,datetime(2022,3,10,9,51,29)).
assert(spo('L-2',label,"Module number of students"),
    48,datetime(2022,3,10,9,52,3)).
assert(spo('H-1',type,'Class'),48,datetime(2022,3,10,9,52,3)).
assert(spo('H-1',subClassOf,'Hazard'),48,datetime(2022,3,10,9,52,3)).
assert(spo('F-1',type,'Class'),48,datetime(2022,3,10,9,52,11)).
assert(spo('F-1',subClassOf,'Fluent'),48,datetime(2022,3,10,9,52,11)).
assert(spo('E-1',type,'Class'),48,datetime(2022,3,10,9,52,11)).
assert(spo('E-1',subClassOf,'Entity'),48,datetime(2022,3,10,9,52,11)).
assert(spo('Q-1',type,'Class'),48,datetime(2022,3,10,9,52,11)).
assert(spo('Q-1',subClassOf,'Quality'),
    48,datetime(2022,3,10,9,52,11)).
assert(spo('F-1',hasBearer,'E-1'),48,datetime(2022,3,10,9,52,11)).
assert(spo('H-1',hasHolding,'F-1'),48,datetime(2022,3,10,9,52,11)).
assert(spo('F-1',hasQualityInSomeSituation,'Q-1'),
    48,datetime(2022,3,10,9,52,11)).
glossary_lookup('Hazard',48,datetime(2022,3,10,9,52,16)).
assert(spo('E-1',label,"Module"),48,datetime(2022,3,10,9,52,39)).
retract(spo('H-1',hasHolding,'F-1'),
    48,datetime(2022,3,10,9,52,40)).
assert(spo('H-1',notHasHolding,'F-1'),48,datetime(2022,3,10,9,52,40)).
assert(spo('Q-1',label,"liked"),48,datetime(2022,3,10,9,52,58)).
assert(spo('H-1',hasPossible,'L-1'),48,datetime(2022,3,10,9,52,58)).
assert(spo('H-2',type,'Class'),48,datetime(2022,3,10,9,53,1)).
assert(spo('H-2',subClassOf,'Hazard'),48,datetime(2022,3,10,9,53,1)).
assert(spo('F-2',type,'Class'),48,datetime(2022,3,10,9,53,5)).
```

```
assert(spo('F-2',subClassOf,'Fluent'),48,datetime(2022,3,10,9,53,5)).
assert(spo('E-2',type,'Class'),48,datetime(2022,3,10,9,53,5)).
assert(spo('E-2',subClassOf,'Entity'),48,datetime(2022,3,10,9,53,5)).
assert(spo('Q-2',type,'Class'),48,datetime(2022,3,10,9,53,6)).
assert(spo('Q-2',subClassOf,'Quality'),
    48,datetime(2022,3,10,9,53,6)).
assert(spo('F-2',hasQualityInSomeSituation,'Q-2'),
    48,datetime(2022,3,10,9,53,6)).
assert(spo('F-2',hasBearer,'E-2'),48,datetime(2022,3,10,9,53,6)).
assert(spo('H-2',hasHolding,'F-2'),48,datetime(2022,3,10,9,53,6)).
assert(spo('E-2',label,"Module leader"),
    48,datetime(2022,3,10,9,53,16)).
retract(spo('H-2',hasHolding,'F-2'),48,datetime(2022,3,10,9,53,25)).
assert(spo('H-2',notHasHolding,'F-2'),48,datetime(2022,3,10,9,53,25)).
assert(spo('Q-2',label,"satisfied with number of students"),
    48,datetime(2022,3,10,9,53,50)).
assert(spo('H-2',hasPossible,'L-2'),48,datetime(2022,3,10,9,53,51)).
request_intervention(48,datetime(2022,3,10,9,54,12)).
focus_step(step2,48,datetime(2022,3,10,9,54,16)).
focus_concept('ControlledProcess',48,datetime(2022,3,10,9,54,16)).
assert(spo('CP-1',type,'Class'),48,datetime(2022,3,10,9,54,19)).
assert(spo('CP-1',subClassOf,'ControlledProcess'),
    48,datetime(2022,3,10,9,54,19)).
focus_subconcept('CP-1',48,datetime(2022,3,10,9,54,22)).
glossary_lookup('ControlledProcess',48,datetime(2022,3,10,9,54,31)).
assert(spo('CP-1',label,"Suggest module"),
    48,datetime(2022,3,10,9,54,54)).
focus_subconcept('CP-1',48,datetime(2022,3,10,9,55,0)).
request_intervention(48,datetime(2022,3,10,9,55,26)).
intervene('Step 2 precedes Step 3',
    not ( logged_assertion(_16478,hasCapability,_16482)
          and fact(_16482,requestsEffect,_16538)
```

```
        and logged_assertion(_16570,subClassOf,'Feedback')),
    1,datetime(2022,3,10,9,55,26)).
focus_concept('Entity',48,datetime(2022,3,10,9,56,4)).
focus_subconcept('E-1',48,datetime(2022,3,10,9,56,7)).
focus_subconcept('E-2',48,datetime(2022,3,10,9,56,18)).
focus_concept('Feedback',48,datetime(2022,3,10,9,56,31)).
focus_concept('ControlAction',48,datetime(2022,3,10,9,56,33)).
focus_concept('Entity',48,datetime(2022,3,10,9,56,34)).
focus_concept('ControlledProcess',48,datetime(2022,3,10,9,56,36)).
focus_step(step1,48,datetime(2022,3,10,9,56,38)).
focus_concept('Loss',48,datetime(2022,3,10,9,56,38)).
focus_concept('ControlledProcess',48,datetime(2022,3,10,9,56,40)).
focus_step(step2,48,datetime(2022,3,10,9,56,40)).
focus_concept('Entity',48,datetime(2022,3,10,9,56,41)).
retract(spo('E-1',label,"Module"),48,datetime(2022,3,10,9,56,52)).
assert(spo('E-1',label,"Student"),48,datetime(2022,3,10,9,56,52)).
focus_concept('ControlledProcess',48,datetime(2022,3,10,9,56,52)).
focus_concept('Entity',48,datetime(2022,3,10,9,56,53)).
focus_step(step1,48,datetime(2022,3,10,9,56,58)).
focus_concept('Loss',48,datetime(2022,3,10,9,56,58)).
retract(spo('Q-1',label,"liked"),48,datetime(2022,3,10,9,57,20)).
assert(spo('Q-1',label,"happy with module"),
    48,datetime(2022,3,10,9,57,20)).
focus_step(step2,48,datetime(2022,3,10,9,57,20)).
focus_concept('ControlledProcess',48,datetime(2022,3,10,9,57,20)).
focus_concept('ControlledProcess',48,datetime(2022,3,10,9,57,23)).
focus_concept('Entity',48,datetime(2022,3,10,9,57,25)).
focus_concept('ControlAction',48,datetime(2022,3,10,9,57,27)).
assert(spo('CA-1',type,'Class'),48,datetime(2022,3,10,9,57,29)).
assert(spo('CA-1',subClassOf,'ControlAction'),
    48,datetime(2022,3,10,9,57,29)).
focus_subconcept('CA-1',48,datetime(2022,3,10,9,57,30)).
```

```
glossary_lookup('ControlAction',48,datetime(2022,3,10,9,57,34)).
focus_step(step1,48,datetime(2022,3,10,9,57,42)).
focus_concept('Loss',48,datetime(2022,3,10,9,57,42)).
focus_concept('ControlledProcess',48,datetime(2022,3,10,9,57,49)).
focus_step(step2,48,datetime(2022,3,10,9,57,49)).
glossary_lookup('ControlAction',48,datetime(2022,3,10,9,58,6)).
assert(spo('CA-1',requestsToNotHold,'F-1'),
    48,datetime(2022,3,10,9,58,31)).
assert(spo('CA-1',label,"Student prefers practical over theoretical"),
    48,datetime(2022,3,10,9,59,14)).
focus_concept('Feedback',48,datetime(2022,3,10,9,59,48)).
assert(spo('FB-1',type,'Class'),48,datetime(2022,3,10,9,59,52)).
assert(spo('FB-1',subClassOf,'Feedback'),
    48,datetime(2022,3,10,9,59,52)).
glossary_lookup('Feedback',48,datetime(2022,3,10,9,59,53)).
focus_subconcept('FB-1',48,datetime(2022,3,10,9,59,56)).
assert(spo('F-3',type,'Class'),48,datetime(2022,3,10,10,0,7)).
assert(spo('F-3',subClassOf,'Fluent'),48,datetime(2022,3,10,10,0,7)).
assert(spo('E-3',type,'Class'),48,datetime(2022,3,10,10,0,7)).
assert(spo('E-3',subClassOf,'Entity'),48,datetime(2022,3,10,10,0,7)).
assert(spo('Q-3',type,'Class'),48,datetime(2022,3,10,10,0,7)).
assert(spo('Q-3',subClassOf,'Quality'),48,datetime(2022,3,10,10,0,7)).
assert(spo('FB-1',recordsFluent,'F-3'),48,datetime(2022,3,10,10,0,7)).
assert(spo('F-3',hasBearer,'E-3'),48,datetime(2022,3,10,10,0,7)).
assert(spo('F-3',hasQualityInSomeSituation,'Q-3'),
    48,datetime(2022,3,10,10,0,7)).
retract(spo('F-3',hasBearer,'E-3'),48,datetime(2022,3,10,10,0,18)).
retract(spo('F-3',hasQualityInSomeSituation,'Q-3'),
    48,datetime(2022,3,10,10,0,18)).
retract(spo('F-3',subClassOf,'Fluent'),
    48,datetime(2022,3,10,10,0,18)).
retract(spo('F-3',type,'Class'),48,datetime(2022,3,10,10,0,18)).
```

```
retract(spo('FB-1',recordsFluent,'F-3'),
    48,datetime(2022,3,10,10,0,18)).
retract(spo('Q-3',subClassOf,'Quality'),
    48,datetime(2022,3,10,10,0,18)).
retract(spo('Q-3',type,'Class'),48,datetime(2022,3,10,10,0,18)).
retract(spo('E-3',subClassOf,'Entity'),
    48,datetime(2022,3,10,10,0,18)).
retract(spo('E-3',type,'Class'),48,datetime(2022,3,10,10,0,18)).
assert(spo('FB-1',label,"Student grades in specific modules"),
    48,datetime(2022,3,10,10,0,42)).
assert(spo('FB-1',recordsFluent,'F-1'),
    48,datetime(2022,3,10,10,0,42)).
focus_concept('Entity',48,datetime(2022,3,10,10,0,50)).
focus_concept('ControlAction',48,datetime(2022,3,10,10,0,53)).
assert(spo('CA-2',type,'Class'),48,datetime(2022,3,10,10,1,20)).
assert(spo('CA-2',subClassOf,'ControlAction'),
    48,datetime(2022,3,10,10,1,20)).
focus_subconcept('CA-2',48,datetime(2022,3,10,10,1,28)).
assert(spo('CA-2',label,"Not enough students enrolled in module"),
    48,datetime(2022,3,10,10,2,3)).
focus_concept('Feedback',48,datetime(2022,3,10,10,2,3)).
assert(spo('FB-2',type,'Class'),48,datetime(2022,3,10,10,2,14)).
assert(spo('FB-2',subClassOf,'Feedback'),
    48,datetime(2022,3,10,10,2,14)).
focus_subconcept('FB-2',48,datetime(2022,3,10,10,2,17)).
assert(spo('FB-2',label,"Number of students enrolled on every
module"),
    48,datetime(2022,3,10,10,2,36)).
request_intervention(48,datetime(2022,3,10,10,2,36)).
focus_step(step3,48,datetime(2022,3,10,10,2,39)).
request_help('Step 2 precedes Step 3',48,datetime(2022,3,10,10,2,54)).
focus_step(step2,48,datetime(2022,3,10,10,3,3)).
```

```
focus_concept('ControlledProcess',48,datetime(2022,3,10,10,3,3)).
focus_concept('Entity',48,datetime(2022,3,10,10,3,4)).
focus_concept('ControlAction',48,datetime(2022,3,10,10,3,4)).
focus_concept('ControlledProcess',48,datetime(2022,3,10,10,3,6)).
focus_concept('Feedback',48,datetime(2022,3,10,10,3,6)).
request_help('Step 2 precedes Step 3',48,datetime(2022,3,10,10,3,9)).
focus_concept('Entity',48,datetime(2022,3,10,10,3,28)).
focus_concept('Feedback',48,datetime(2022,3,10,10,3,30)).
focus_concept('Entity',48,datetime(2022,3,10,10,3,34)).
focus_subconcept(null,48,datetime(2022,3,10,10,4,3)).
assert(spo('E-1',hasCapability,'CA-1'),
    48,datetime(2022,3,10,10,4,56)).
focus_subconcept('CA-1',48,datetime(2022,3,10,10,5,10)).
assert(spo('E-2',informedBy,'FB-2'),48,datetime(2022,3,10,10,5,26)).
request_intervention(48,datetime(2022,3,10,10,5,33)).
intervene('Check that every controlled physical process is \
    controlled by one or more controllers',
    logged_assertion('CP-1',subClassOf,'ControlledProcess')
    and not ( logged_assertion('CP-1',hasSubject,_8594)
            and logged_assertion(_8646,hasBearer,_8594)
            and fact(_8702,requestsEffect,_8646)
            and logged_assertion(_8738,hasCapability,_8702)),
    1,datetime(2022,3,10,10,5,33)).
request_intervention(48,datetime(2022,3,10,10,5,37)).
request_intervention(48,datetime(2022,3,10,10,5,43)).
assert(spo('FB-2',recordsFluent,'F-2'),48,datetime(2022,3,10,10,7,5)).
focus_concept('Feedback',48,datetime(2022,3,10,10,8,1)).
assert(spo('FB-3',type,'Class'),48,datetime(2022,3,10,10,8,3)).
assert(spo('FB-3',subClassOf,'Feedback'),
    48,datetime(2022,3,10,10,8,3)).
focus_subconcept('FB-3',48,datetime(2022,3,10,10,8,5)).
assert(spo('FB-3',label,"Contents of module"),
```

```
    48,datetime(2022,3,10,10,8,16)).
assert(spo('FB-3',recordsFluent,'F-1'),
    48,datetime(2022,3,10,10,8,22)).
focus_concept('Entity',48,datetime(2022,3,10,10,8,23)).
assert(spo('E-1',informedBy,'FB-3'),48,datetime(2022,3,10,10,8,27)).
focus_concept('Feedback',48,datetime(2022,3,10,10,8,44)).
focus_concept('Entity',48,datetime(2022,3,10,10,8,51)).
focus_concept('ControlledProcess',48,datetime(2022,3,10,10,8,52)).
focus_concept('ControlAction',48,datetime(2022,3,10,10,9,7)).
focus_concept('Entity',48,datetime(2022,3,10,10,9,9)).
focus_concept('ControlledProcess',48,datetime(2022,3,10,10,9,25)).
assert(spo('CP-1',hasSubject,'E-2'),48,datetime(2022,3,10,10,9,27)).
focus_concept('Entity',48,datetime(2022,3,10,10,9,44)).
request_intervention(48,datetime(2022,3,10,10,10,19)).
request_intervention(48,datetime(2022,3,10,10,10,22)).
request_intervention(48,datetime(2022,3,10,10,10,26)).
focus_step(step3,48,datetime(2022,3,10,10,10,38)).
glossary_lookup('UCA',48,datetime(2022,3,10,10,10,41)).
assert(spo('CA-1',subClassOf,
            'NotProvidingPotentiallyHazardousControlAction'),
                48,datetime(2022,3,10,10,11,18)).
assert(spo('CA-2',subClassOf,
            'NotProvidingPotentiallyHazardousControlAction'),
                48,datetime(2022,3,10,10,11,23)).
assert(spo('CA-1',notProvidingPotentiallyLeadsTo,'H-1'),
    48,datetime(2022,3,10,10,11,43)).
assert(spo('CA-2',notProvidingPotentiallyLeadsTo,'H-2'),
    48,datetime(2022,3,10,10,11,53)).
request_intervention(48,datetime(2022,3,10,10,12,34)).
focus_step(step4,48,datetime(2022,3,10,10,12,40)).
focus_step(step4,48,datetime(2022,3,10,10,12,44)).
focus_step(step3,48,datetime(2022,3,10,10,12,44)).
```

```
focus_step(step4,48,datetime(2022,3,10,10,12,45)).
```