Refinement Patterns for Contractual Statecharts

Dr. Gerald Luettgen and Dr. Richard Paige

Summary

The strategic goal of this project is to improve the theoretical bases and corresponding tool support for the design languages and methodologies that are widely used for building avionics and aerospace systems. Existing languages, particularly the popular Stateflow statecharts, lack in expressiveness and tool support for refinement-based designs as practised by engineers. In particular, they have no support for declarative styles of specification used at early design stages, and their semantics do not entirely benefit from the existing wealth of research on concurrent systems and semantics in component–based design. The specific novel contributions of this proposed research project are:

1. The definition of the syntax and semantics of an extension of Stateflow statecharts, *Contractual Statecharts*, which supports developers in writing models in mixed declarative and operational specification styles;
2. A *refinement relation* for component–based, stepwise design that allows for trading off operational for declarative content in models;
3. Supporting simulation and model checking tools, which shall be developed and implemented so as to integrate with the industry-standard Stateflow tool, to enable engineers to analyse their mixed–style models;
4. A set of *refinement patterns* capturing standard rules expressing how to translate between declarative and operational styles of specification, and a tool assisting in the application of refinement patterns.

The project will be driven by industrial case studies and, where necessary, will involve feedback from the industrial partner on the utility of the language, supporting tools and patterns.

1 Previous Research Track Record

Dr. Gerald Luettgen. GL joined the Programming Languages and Systems group at the University of York as Senior Lecturer in 2003. He completed his PhD in Computer Science at the University of Passau, Germany, in 1998, before becoming a Research Scientist at ICASE / NASA Langley Research Center.

GL's expertise is in the fields of concurrency and automated verification. He was the PI of the EPSRC-funded research project "Type Analysis for Component-based Real-time Programming" (GR/M99637/01) which developed a process-algebraic, bisimulation-based reactive-type theory and supporting tools for applications in digital signal processing [13]. Currently GL is the PI of the EPSRC-funded research project "New-generation Symbolic Model Checkers for Verifying Asynchronous Systems" (GR/S86211/01) which explores novel model-checking technologies that exploit the event locality inherent in asynchronous systems in order to increase verification efficiency [3]. GL's experience gained within both projects will particularly help in the proposed project with designing a component-based theory that is usable in practice, and with providing efficient automated tool support.

Of most relevance to this project is GL's expertise with formalisms combining operational and assertional specification styles, and with statecharts. While conducting research at NASA Langley Research Center, GL was a co-investigator of the NSF-funded research project "Heterogeneous Specification Formalisms for Reactive Systems" (NSF/9988489) which defined a uniform, refinement-based semantics for combining the process algebra CCS and the temporal logic LTL [5]. The difference to the proposed project is that CCS+LTL is a much more expressive language than the proposed Contractual Statecharts language, as it can additionally express fairness and unbounded liveness properties. However, it has a very complex semantics that is not engineer-friendly and does not lend itself readily to efficient tool support.

In recent years, GL's research on the process-algebraic and model-theoretic semantics of statecharts and Esterel has gained international recognition as it provided insights into how these synchronous languages are related and established novel full-abstraction results [9]. Last, but not least, GL's experience with the design of avionics software at NASA means that he is familiar with the ad-hoc design methodologies employed by avionics engineers and with the domain of the case studies provided by our industrial partner.

Dr. Richard Paige. RP joined the High-Integrity Systems Engineering group at the University of York as Lecturer in 2001. He completed his PhD in Computer Science at the University of Toronto in 1997.

RP is currently the PI of the EPSRC-funded research project "Agile Development of High-Integrity Grid Middleware" (GR/S64226/01) which is developing a software engineering methodology for producing more dependable Grid systems. RP is also an investigator in the EU IST project MODELWARE which is the largest software engineering project currently run by the European Commission. His contribution to this project is the development of industrially applicable behavioural semantics for UML-like languages and formal analytic techniques for determining whether UML-like specifications are consistent. Since 2002, RP has been a strand leader in the Defence and Aerospace Research Partnership (DARP) for High-Integrity Real-Time Systems, where he is leading research on model-based systems engineering. His experience within these projects will help in the proposed project in the development of language semantics that are usable by engineers, and in his experience in building dependable computing systems.

Of particular relevance to the proposed project is RP's work on UML semantics within ModelWare [14]. This work developed an action semantics for the core of UML (the Meta-Object Facility) which was usable by MODELWARE's industrial partners (e.g., Thales, Adaptive, France Telecom) and the tools they used in day-to-day engineering. His work on agile development for Grids is also relevant as it is producing principles and practices, usable by engineers, for improving the dependability of complex and, in this case, distributed systems [2].

RP's recent work on methodologies for meta-modelling in engineering language design has seen international recognition for distilling the principles and practices in constructing modelling languages [15]. Some of this work is being used in the DARP project where the domain of focus is avionics and aerospace systems; he is therefore familiar and experienced with the domain of interest to our industrial collaborators.

Institutional expertise and support. The RAE 6*–rated Computer Science Department at the University of York has several internationally renowned research groups with expertise in the fields of software engineering and formal methods, in particular regarding their application to High–Integrity Systems Engineering, Programming Languages and Systems and Real–Time Systems which are of interest to this research proposal. Current research in the Department is funded by grants and contracts totalling £5 million from the EPSRC, the European Commission, other government departments and industry.

The Department is home to the BAE Systems Dependable Computing Centre which focuses on dependable avionics systems, and which has worked on statecharts for many years. It also houses the Rolls-Royce University Technology Centre which undertake basic research and technology transfer of academic research into industrial practice, particularly for aircraft engine systems. These industrial centres are part of the High-Integrity Systems Engineering (HISE) group, led by Prof. John McDermid, and which is made up of approximately 50 researchers experienced with all aspects of engineering high-integrity, and particularly aerospace, systems. Within HISE there is substantial expertise on the use of statecharts and tool support for the widely used Stateflow dialect of statecharts [18], particularly by Dr. Ian Toyn and Dr. Andy Galloway.

The Programming Languages and Systems group, led by Prof. Colin Runciman, has substantial expertise in the fields of language design, concurrency theory, formal semantics, automated verification and tool support. There is also expertise in the Department on embedded systems, particularly from the Real-Time Systems group.

2 Description of Proposed Research

2.1 Background and Related Work

Background. Avionics and aerospace systems are complex, involving computer hardware, software and mechanical/hydraulic devices (e.g., ailerons). Engineers of such systems are faced with engineering deadlines, hard physical constraints (e.g., the architectural design of the airframe), safety constraints, and unchangeable hardware constraints (e.g., the maximum processor speed). A variety of development techniques are applied in order to produce high-quality embedded real-time software for avionics and aerospace systems, but the state-of-the-practice is lacking in a number of respects: (i) an explicit *software architecture* is not regularly constructed, making it difficult to identify reusable components and to manage change due to modified customer requirements; (ii) the *models* that are produced during development often are at odds with engineering judgement in that they can neither adequately express engineering solutions nor accurately reflect engineering design discipline; (iii) there is no formal basis for the *refinement* process employed when step-by-step refining the initial, largely declarative specifications to the final, operational designs; and (iv) *tool support* for analysing the models that are produced is remarkably limited and mostly comprises of a drawing tool and a simulator. This last point is critical: one of the benefits of modelling early in the development process is to catch mistakes and omissions before they become embedded in code. Without better analytic techniques for models for aerospace and avionics systems, the state-of-the-practice will be difficult to improve.

This project aims to address this last point, by linking together established and novel theoretical ideas in refinement-

based designs with practical application to aerospace and avionics systems engineering, in the domain of *statecharts* technology. This will be carried out by extending Stateflow statecharts with lightweight *contracts* and enabling statecharts to be used in combination with more declarative styles of specification. Our research will facilitate the formal study of the refinement technologies used by engineers, alleviate some of the practical difficulties associated with extracting statecharts from existing systems, and make it easier to *analyse* practical statecharts models.

Related work. Regarding semantic foundations of multi-paradigm design languages, previous work by GL with Rance Cleaveland [5] studied a combination of the process algebra CCS and the temporal logic LTL and based the refinement preorder on the theory of testing rather than bisimulation. The mixed language CCS+LTL is clearly targeted at concurrency theoreticians and not engineers, and no aim at axiomatising the refinement preorder or providing refinement patterns was made.

Regarding extensions of statecharts, Galloway and Toyn [6, 8] have extended the Stateflow language to include a simple annotation language on states, which is used in the formal validation of Statecharts. The annotations could be considered as simple contracts, however they only cover a very restrictive subset of Stateflow behaviour (e.g., no support for events, transition actions, and-states). Moreover there is no support structural refinement at the chart level. Our proposed work differs in that it (i) supports a mixture of specification styles, as well as a contract language that can express more than simple, propositional assumptions; (ii) allows for more flexibility in placing contracts, such as placing contracts on transitions, e.g., when requiring a combination of states to be active or inactive in order for a given transition to be enabled; (iii) is equipped with a theory and tool support for refinement checking.

In another line of research, Sowmya and Ramesh have extended statecharts with Lamport's temporal logic TLA [16]. In contrast to what we propose, they allow declarative information to be attached to states only and not also to transitions, although their declarative language TLA is more expressive than our contract language. Their semantics is based on logics, so that refinement between designs reduces to logical implication. However, they neither focus on component-based refinement, nor do they propose refinement patterns.

While our concept of refinement patterns in this context is novel, it has some similarity to design patterns. Design patterns have also been studied for statecharts [19], with the aim of facilitating the reuse of statechart implementations. Refinement strategies in the context of refinement calculi or programming methodologies [17] have also some similarity to what we are proposing; the main differences are the context (typically data transformation or algorithmic refinement [17]), the single style of specification (usually declarative) and the scope (typically strategies focus on behavioural refinement). We intend to look carefully at this work to help motivate additional refinement patterns of our own. More distantly related is earlier work on refinement calculi and tools, e.g., [1], which attempted to provide support and a collection of strategies for refining specifications into programs.

A key problem with expressing refinement patterns is the form in which their specification is made. Clark et al. [4] proposed a component-based form of pattern specification with "holes", wherein missing units of functionality (e.g., classes and objects) and constraints on instantiation (e.g., preconditions on when the pattern can be applied) are encapsulated within one UML-like component. We anticipate using a variant of this approach in cataloguing our own refinement patterns; the approach in [4] has not previously been applied in this domain.

Last, but not least, it needs to be emphasised that our notion of refinement is not concerned with safety as in Safety Engineering, since safety does not refine. Instead, we consider the refinement of system specifications and designs, by successively trading off concrete, operational content for abstract, declarative content. However, our concept of refinement might help in arguing safety cases, but investigating this is not in the scope of the proposed project.

2.2 Research Programme and Methodology

Aims and objectives. Software engineers often rely on an ad-hoc, component-based design methodology that step-by-step refines abstract *reactive–systems designs* given in a mixed operational and declarative notation into concrete, fully-operational designs. The project's aims are to put this methodology on a formal footing via so-called *refinement patterns*, and to support the methodology by the development of state-of-the-art tools that integrate with the popular *Matlab/Stateflow* design tool [18]. These aims will be achieved by pursuing the following objectives:

I. To develop a novel and elegant language, *Contractual Statecharts*, that combines the operational statecharts language with contracts (cf. Eiffel and Spark Ada) for declaratively specifying behaviour, and to implement this language by extending the Matlab/Stateflow tool with a facility for specifying contracts;

IIa. To provide a structural operational semantics for Contractual Statecharts on the basis of *extended I/O automata*, and implement a simulator for the Stateflow extension which executes this semantics;

IIb. To develop a compositional refinement preorder for Contractual Statecharts which reflects the component-based design methodology;

IIIa. To establish the concept of *refinement pattern*, to provide example refinement patterns, and to develop tool support for applying refinement patterns;

IIIb. To provide model-checking tool support for the refinement preorder (in the finite-state case);

IV. To conduct case studies, provided by our industrial partner, that apply the developed refinement patterns and tools to the design of avionics software.

The outcome of the proposed research will be (i) a new language, *Contractual Statecharts*, permitting the description of mixed operational and declarative designs of reactive systems, (ii) a set of refinement patterns for the stepwise refinement of abstract, declarative designs into concrete, operational designs and (iii) a suite of tools consisting of a simulator, a model checker and a refinement pattern applier, all of which interface to the Matlab/Stateflow tool set. Using these results, engineers will be able to profit from the formal tool support underpinning one of their most practised design methodologies and to share and reuse refinement patterns among different projects.

Methodology. At the centre of the project is a novel design language for reactive–systems software, particularly avionics software, that combines ideas of *statecharts*, such as supported by the commercial tool *Stateflow* [18] and *contracts* [11], such as employed in the programming languages Eiffel and Spark Ada. An extensive study of case studies provided by our industrial partner and textbook examples will enable us to identify the core subset of Stateflow to be supported, as well as the contract language to be designed. The contract language shall be clear and unambiguous, permit an elegant definition of its semantics and must allow for specifying safety properties and bounded liveness, including pre-/post-conditions and system invariants. Our novel design language shall enable engineers to attach contracts to both statecharts states and transitions, whence we will refer to it as *Contractual Statecharts*. To support this language within Matlab/Stateflow requires us to properly extend the tool's graphical interface and to modify the internal textual representation of Stateflow diagrams.

The semantics of Contractual Statecharts will be based on an extension of Lynch and Tuttle's *I/O automata* [10], where states may be annotated with actions and variables, and with information whether the choice between outgoing transitions of a given state is a nondeterministic one, or a logical one representing disjunction. Specific states encoding blocked design (deadlock), universal design (true) and unimplementable design (false) must also be distinguished. This semantics shall be complemented by a simulator tool for Contractual Statecharts, which will allow us to validate whether our semantics conforms to Stateflow and the desired contract semantics.

Moreover, a refinement preorder will be developed that formalises the engineering practice of step-by-step refining abstract, declarative designs into concrete, operational designs, by successively replacing contractual contracts specifying the behaviour of a component by a statechart satisfying this prescribed behaviour. Such component-based reasoning demands of our refinement preorder to be compositional. As our focus of application is on control software, we suggest the refinement relation to be found on Park and Milner's notion of (bi-)simulation. For Contractual Statecharts representing finite-state systems, such a refinement relation will be decidable, and we can use existing ideas of *characteristic formulae* for behavioural relations [7] to reduce the decision problem to a satisfiability problem in propositional logic. Accordingly, a model checker shall be developed that interfaces to a powerful SAT-solver [12].

Many engineers would want to stepwise refine their designs according to "refinement patterns" that have proved successful in the past. Although being implicitly used for decades, refinement patterns are not a formally established concept. Refinement patterns are pairs of parameterised designs, i.e., designs including unspecified or under-specified components, whose concrete instantiations are all related according to our refinement preorder. Much care will be given to identify both structural and behavioural refinement patterns that are commonly used in reactive-systems design and particularly in avionics. Tool support shall be provided for applying these refinement patterns.

Our language of Contractual Statecharts, our refinement theory and patterns, and our tool support will be exercised and evaluated by means of two realistically-sized case studies provided by our industrial partner.

Originality and timeliness. Engineers today are already using multi-paradigm design methodologies that mix operational and declarative styles. To specify operational behaviour, engineers typically resort to a dialect of statecharts, such as implemented in the popular Matlab/Stateflow tool set. To specify declarative behaviour, engineers often provide pre-/post-conditions and invariants within design documents and implementation languages such as Eiffel or Spark Ada. However, multi-paradigm design methodologies are still used ad-hoc, and neither have a sound formal underpinning nor do they support the validation of design steps. Research has instead focused on design patterns to improve reusability, reduce design time and increase design quality; these patterns focus on transforming designs into implementations, but not on refining high-level declarative designs into low-level operational designs.

These observations lead immediately to our ideas of Contractual Statecharts for specifying multi-paradigm designs, and of refinement patterns for trading-off declarative content for operational content in a component-based, step-by-step manner. Our theory underlying Contractual Statecharts and refinement patterns will put the described multi-paradigm design methodology on a sound footing and enable the provision of tool support. In contrast to other approaches to refinement–based system design, our approach does not require engineers to conduct refinement proofs themselves: they can either rely on already proved refinement patterns or, if the provided pattern repository does not suffice for a particular application at hand, suggest a refinement step of their own that will be automatically checked for correctness. Although our focus is thus not on generality but on practicality, the proposed project will also provide groundwork for unifying different design methodologies in more general ways in the future.

The timeliness of our proposal is also evidenced by related research currently in preparation overseas, most notably at the group of Prof. Willem-Paul de Roever at the University of Kiel, Germany. Its researchers, in particular Dr. Harald Fecher, wish to investigate top-down refinements of statecharts based on under- and over-approximations of execution traces. To specify such approximations, they intend to extend statecharts by temporal logics so as to be able to express general liveness and fairness properties. In contrast, our approach will be less general – as it considers only contracts expressing safety and bounded liveness properties – but additionally focuses on refinement patterns and emphasises tool support. We plan to collaborate with this group regarding the semantic foundations of temporal-logic extensions of statecharts and the definition of open interfaces for tool support.

Programme of work. The aims and objectives described above will be accomplished via a programme of work constituting four inter-related phases. Informally, these are (I) Defining language support for Contractual Statecharts; (II) Defining and validating the semantics of the language; (III) Developing refinement support; and (IV) Evaluating the language and refinement techniques via industrially valuable case studies. The project is jump-started by first analysing existing industrial case studies and textbook examples, in order to inform Phase (I).

(I) Language support. The first phase is to define an integrated language for Contractual Statecharts, where we shall build the language on a core of the Stateflow dialect of statecharts. Stateflow is used widely by our industrial partner and in the aerospace, avionics and defence domains; case studies provided by our industrial partner will guide us in identifying an appropriate core of Stateflow. Restricting ourselves to such a core is necessary, as Stateflow has many features, such as the 12-o'clock rule for resolving nondeterminism, that make it difficult to provide a clean semantics. Fortunately, most of these features are not used for designing safety-critical software. We anticipate that the Stateflow core must include actions (activities), history junctions (history states), connective junctions (conditional states), supertransitions (interlevel transitions), local variables and guards (conditions), since this is a subset used by domain experts, is computationally expressive and contains no substantial redundancy. Additional constructs will be considered through careful analysis of case studies supplied by the industrial partner, and by examining textbook examples such as the classic elevator. It is important to note that this project focuses on statecharts, whence extending our research results from Stateflow to include Simulink block is future work.

To the identified statecharts core will be added a simple language of contracts. The contracts shall allow us to capture and reason about pre-/postconditions and invariants over environment and generated events, action and state names, and variables as well as bounded temporal behaviour, e.g., "within k steps", "along all reaction steps", "in the next reaction step". Thus, we envision using a linear-time temporal logic. This would be useful for our industrial partner since aerospace engineers anecdotally think about linear time in a declarative style (as supported by our contract language) and branching time in an operational style (as supported by our statecharts language).

To syntactically integrate contracts with statecharts, both a visual and textual dialect for the integrated language will be produced. This will allow engineers to switch easily between declarative and operational styles of specification as necessitated by their engineering judgement. Textual representations of statecharts are often considered a tool implementation issue, but engineers frequently prefer to express design textually.

The result of this will be an integrated Contractual Statechart language. There are several challenges to be addressed: identifying the exact list of language features, starting from the core set described above; standardising on a visual representation; defining a textual dialect for the language. An overarching issue here is the notion of *interfaces* as the language shall be component-based. While statecharts naturally support components, they are not entirely self-contained due to, e.g., inter-level transitions. To support compositional reasoning, states through which inter-level transitions are allowed to pass will need to be equipped with a proper interface.

(II) Language semantics. This phase concentrates on producing a sound semantics for the integrated language defined in Phase (I). The main requirement for the language to be industrially useful is that the semantics supports *compositional* reasoning; otherwise, the language and its supporting tools will simply not scale.

Statecharts' step semantics can be given in terms of I/O automata [10], appropriately extended to deal with features such as actions and history states. Our contract language can also be given an operational semantics using structural-operational-semantics (SOS) rules, but the underlying automata model must be able to distinguish between logical disjunction and process choice, as well as the extreme contracts *true* and *false*. Hence, part of this work will be to suitably extend the I/O automata model. The challenge here is to produce a concise semantics so as to facilitate both a clean implementation and simple proofs about system properties.

A simulator for Contractual Statecharts will also be developed – partly because engineers routinely employ simulation for systems analysis and partly because only a simulator will allow us to validate whether our semantics is compatible to the original semantics of Stateflow. This activity is supported by our collaborator Prof. Rance Cleaveland who is an expert in the undocumented semantics of Stateflow. He is Executive Director of the Fraunhofer Institute for Experimental Software Engineering at the University of Maryland and CEO of Reactive-Systems, Inc. (www.reactive-systems.com) which sells a test-case generator and a simulator for Simulink/Stateflow.

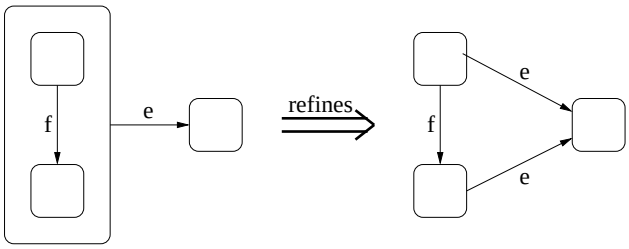
In addition, a refinement semantics will be given, based on a preorder inspired by (bi-)simulation. This will be

defined so that a concrete design refines an abstract design if both are essentially bisimulation equivalent but some disjunctive choice in the abstract design is resolved in the concrete design. Basing the preorder on bisimulation is crucial in this application domain. The preorder must be compositional for the Contractual Statecharts language, as the refinement process must be done in a step-wise and component-wise fashion for it to be scalable, and because our refinement patterns demand an open-systems rather than a closed-systems view.

The challenges with this part of the work are to fully explore the design choices for the refinement preorder, making sure that compositionality is preserved while matching designer intuition; the advice of our industrial collaborators will be vital here. It is likely that there will be a feedback loop between the definition of the preorder and the definition of the aforementioned SOS-rules in order to reach a stable and useful semantics.

(III) Developing refinement support. The third phase of the project is to develop industrially applicable support for the refinement method developed in Phase (II). This will be accomplished by defining a suitable notion of *refinement pattern* and by developing a *model checker* for the refinement preorder.

Refinement patterns in the context of Contractual Statecharts is a new concept. Inspired partly by design patterns and by refinement strategies in the context of program refinement calculi, as well as axiomatisations of preorders in process algebra, refinement patterns provide a mechanistic way to promote easier use of the refinement technique. A refinement pattern is a pair of abstract and concrete templates, so that when a particular Contractual Statechart under investigation matches the abstract template (i.e., is an instantiation of the abstract template), then it may be refined by the appropriate instantiation of the concrete template. A simple example is given in below.



We term the pattern shown on the left *structural*, as one structure is refined by a behaviourally equivalent structure. Another example would be a refinement pattern for replacing declarative (contractual) specifications with operational (statecharts) specifications. Consider a set of three switches specified as statechart, with the contract that only one switch is on at any given time. One *behavioural* refinement pattern would be to refine the Contractual Statecharts specification of such behaviour with a statechart in which the contract is realised via event broadcasting.

We will identify a small set of structural and behavioural refinement patterns by revisiting the case studies that we considered for defining the Contractual Statecharts language in Phase (I). We need to identify refinement patterns that are industrially relevant and which have the potential for mechanisation via automatic identification. For this we will closely consult our industrial collaborators who are already applying ad-hoc refinement strategies in practice as they make use of Stateflow. A challenge here is to define a precise language for specifying refinement patterns for Contractual Statecharts. Such patterns can, informally, be thought of as statecharts with "holes", together with a constraint on what can be placed in these holes; these constraints ensure that any instantiation of the pattern is syntactically and semantically valid. We envision exploiting the ideas of the template specification language defined by Clark et al. [4] as part of their submission for UML 2.0. Moreover, we will prove our refinement patterns correct, i.e., all valid instantiations of abstract and concrete template of a given pattern are related by our preorder.

Application of the refinement patterns shall be supported with a tool so that (i) an engineer can highlight the component of the Contractual Statechart under investigation, select a pattern, and (ii) the tool will verify that the highlighted component is an instance of the abstract template of the selected pattern, and if so, (iii) the tool will replace the abstract template with an instance of the concrete template. This requires the tool to be able to hold a repository of patterns, to which further patterns shall be easy to add.

Many refinement patterns used in practice are not specific for a given system design, but rather a specific application domain. This is why we primarily focus on embedded avionics and aerospace software. However, some patterns may be very specific to the application under consideration, so that the engineer must be able to manually construct a refinement of a given system. Nevertheless, the engineer should be supported in proving that the proposed refinement is correct. This leads to the issue of developing a *decision procedure* for our refinement relation.

If a Contractual Statechart under investigation happens to be finite state, then our refinement preorder should be decidable and efficiently computable. Bisimulation-based preorders are normally efficiently computable via partition-refinement algorithms, e.g., as is done in the Concurrency Workbench. The challenge here is that our preorder is not defined on simple labelled transition systems but on the more expressive extended I/O-automata. One successful technique for checking bounded temporal properties occurring in Contractual Statecharts is to translate into a satisfiability problem on propositional formulae and to use a SAT-solver for bounded model checking, thereby helping us to avoid the state-space explosion problem. This route shall be followed in this project and there is hope that structural properties of statecharts (e.g., guards of transitions being mutually exclusive) can be exploited to make the SAT-based checking more efficient. Implementing a model checker entails (i) conducting a static analysis to determine whether the behaviour of a Contractual Statechart under consideration is finite state and (ii) implementing

a translation of the refinement preorder/partitioning algorithm in SAT. While the latter is novel, there is work on characteristic formulae for pre-bisimulation that we can adapt [7]. As the contracts that are part of our language can only capture safety and bounded liveness properties, the characteristic formulae will also be able to be expressed as propositional formulae. In addition, it shall be investigated how best to relay counterexamples to engineers in case the model checker detects a violation of our refinement preorder.

(IV) Case studies and evaluation. The work outlined in the previous sections will be driven by case studies from our collaborators; these will be used to evaluate the language, its semantics, the refinement patterns and the tool support. We envision taking two industrial case studies and re-expressing and re-assessing them once our tool support is available. This evaluation will be done together with our partners, to ensure that the tools delivered provide value to engineers. To validate that the languages and tools we deliver are general and do not work only on the case studies of our collaborators, we will introduce a third case study at this stage.

Workplan. A detailed diagrammatic workplan is attached and comprises the abovementioned four phases. The contents of each phase's work packages, the packages' timing and the distribution of work between the two full-time researchers employed on the project, one RA and one project student, can all be found in this workplan.

Although our research ambitions are high, the risks are well calculated. Should early work packages take significantly longer than anticipated, we propose not to develop the model checker, i.e., deleting WPs III.4–6. While having a model checker is highly desirable, we could always conduct our case studies using refinement patterns only. Should delays occur at late project stages, we will conduct only two of the three proposed case studies, see WP IV.1 and IV.2. In either case, we would appropriately re-assign the work packages between RA and PhD student in order to re-balance the workload and to guarantee that the PhD student conducts PhD-level research.

2.3 Relevance to Beneficiaries

The software engineering community will benefit from the formal underpinning of a widely but informally used component-based design methodology, which mixes ideas of operational and declarative specification languages and refines system designs by trading off declarative for operational behaviour. The project will conceptually as well as pragmatically establish refinement patterns as a technique for systems design, thus complementing well-known design patterns for systems implementation.

The formal methods and automated verification communities will benefit from advancing the theory of such mixed design languages and, in particular, of component-based refinement relations, as well as from our experiences in employing SAT-solving technologies in deciding refinement relations for finite-state designs.

The aerospace, avionics and defence industries will benefit from the access to and the evaluation of novel tools that support the design of typical systems considered by these companies. As the tools to be developed are integrated with the Matlab/Stateflow design tool already used in the design process employed in these industries, they will not interfere with the engineers' current design flow but rather improve the quality of designs. Particular beneficiaries here are our industrial partner, as they provide the case studies that will be studied within the research project; they profit from us carrying out a feasibility study of the developed technology within their application domain.

2.4 Dissemination and Exploitation

The dissemination of our research results will proceed along two routes. The first, classical route consists of presentations at international conferences and publications in renowned journals. The following lists the main conferences and journals of interest to us, organised by research topic of relevance to the project:

Topic	Conferences	Journals
Language/simulation	FSE	IEEE TSE
Semantics	CONCUR, ICALP	FACS, Inform. & Comp.
Refinement patterns	FSE, FM	IEEE TSE, ACM TOCL, STTT
Model checking	TACAS, CAV, AVOCS	FMSD, FACS, STTT
Case studies	ISSC, SafeComp	IEEE TSE, ACM TOSM, STTT

In addition to presentations at conferences we also plan presentations at our industrial partner, comprising of formal presentations for the management and engineering presentations for the potential users of our technology.

The second dissemination route will make our developed software tools, the produced user manuals and the conducted case studies available via the Internet. This will further increase the visibility of our research and encourage the uptake of our technology by both the software engineering community and the embedded-systems community.

The commercial exploitation of our research results and tools will be evaluated together with our industrial partner, as well as tool vendors such as HIS, at the end of the project. We envision that our refinement-based design technology and tools are highly applicable to a wide class of avionics control systems. This is because the technology puts

current ad-hoc design practices on a sound footing and because the tools are designed to integrate well with the work flow currently practised in the avionics and defence industries, which already use Stateflow in their tool chain.

References

[1] D. Carrington, I. Hayes, R. Nickson, G. Watson, and J. Welsh. A tool for developing correct programs by refinement. In *BCS Refinement Workshop*, eWIC, 1996.

[2] H. Chivers, R. Paige, and X. Ge. Agile security via an incremental security architecture. In *Extreme Programming*, vol. 3556 of *LNCS*, pp. 57–65, 2005.

[3] G. Ciardo, G. Lüttgen, and R. Siminiceanu. An efficient iteration strategy for symbolic state-space generation. In *TACAS*, vol. 2031 of *LNCS*, pp. 328–342, 2001.

[4] T. Clark, A. Evans, P. Sammut, and J. Willans. *Applied Metamodelling*. www.xactium.com, 2004.

[5] R. Cleaveland and G. Lüttgen. A logical process calculus. In *EXPRESS*, vol. 68,2 of *ENTCS*, 2002.

[6] A. Galloway and I. Toyn. Proving properties of Stateflow models using ISO Standard Z and CADiZ. In *ZB-2005*, vol. 3455 of *LNCS*, pp. 104–123, 2005.

[7] A. Ingolfsdottir and B. Steffen. Characteristic formulae. *Information and Control*, 110(1):149–163, 1994.

[8] F. Iwu, A. Galloway, I. Toyn, and J. McDermid. Practical formal specification for embedded control systems. In *INCOM*, 2004.

[9] G. Lüttgen and M. Mendler. The intuitionism behind Statecharts steps. *ACM TOCL*, 3(1):1–41, 2002.

[10] N. Lynch and M. Tuttle. An introduction to I/O automata. *CWI-Quarterly*, 2(3):219–246, 1989.

[11] B. Meyer. Applying design by contract. *IEEE Computer*, 25(10):40–51, 1992.

[12] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *DAC*. IEEE/ACM, 2001.

[13] B. Norton, G. Lüttgen, and M. Mendler. A semantic theory for synchronous component-based design. In *CONCUR*, vol. 2761 of *LNCS*, pp. 461–476, 2003.

[14] R. Paige, D. Kolovos, and F. Polack. Refinement via consistency checking in MDA. In *REFINE*, ENTCS, 2005.

[15] R. Paige, J. Ostroff, and P. Brooke. Theorem proving for view consistency checking. *L'Objet*, 9(4), 2003.

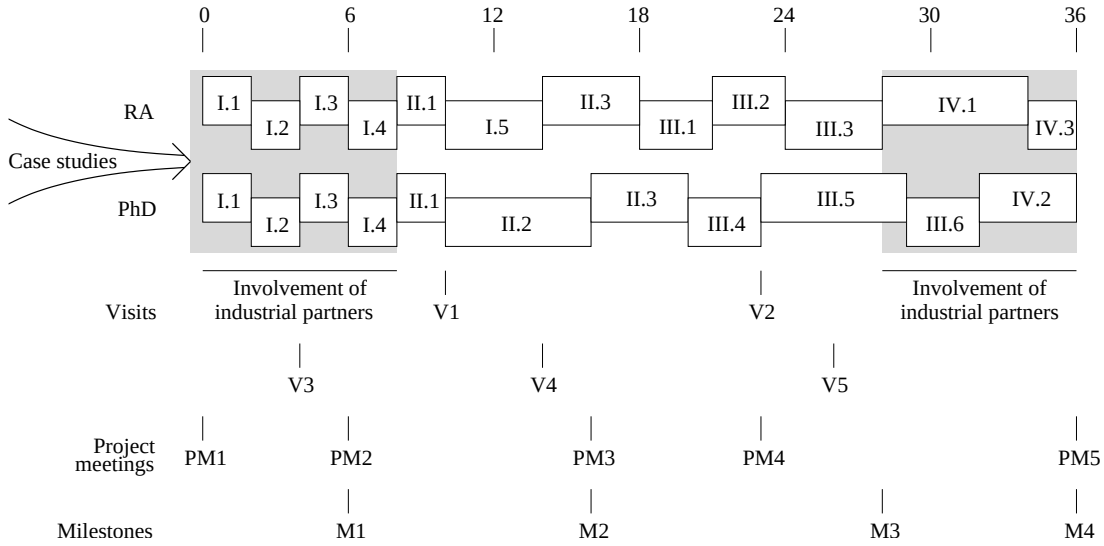
[16] A. Sowmya and S. Ramesh. Extending Statecharts with temporal logic. *IEEE TSE*, 24(3):216–231, 1998.

[17] S. Stepney, F. Polack, and I. Toyn. Patterns to guide practical refactoring. In *ZB-2003*, vol. 2651 of *LNCS*, pp. 20–39, 2003.

[18] The MathWorks, Inc. Matlab/Simulink/Stateflow tool suite. www.mathworks.com.

[19] S. Yacoub and H. Ammar. A pattern language of state charts. In *PloP '98*, vol. WUCS-98-25 of *Techn. Rep. Series, Washington Univ., Dept. Computer Science*, 1998.

Appendix: Diagrammatic Workplan



Work packages

Phase I: Defining the Contractual Statecharts language
WP I.1: Get familiarised with case studies and textbook examples
WP I.2: Identify suitable subset of Stateflow
WP I.3: Identify suitable contract language
WP I.4: Define combined Stateflow+contract language
WP I.5: Implement combined language in Stateflow

Phase II: Defining and validating the semantics of Contractual Statecharts
WP II.1: Define semantics of Contractual Statecharts in terms of extended I/O automata
WP II.2: Develop bisimulation-based refinement relation and prove compositionality
WP II.3: Write a simulator for Contractual Statecharts and validate semantics

Phase III: Developing refinement support
WP III.1: Identify structural refinement patterns
WP III.2: Identify behavioural refinement patterns
WP III.3: Provide tool support for applying refinement patterns
WP III.4: Identify and implement finiteness checks of Contractual Statecharts designs
WP III.5: Develop and implement model-checking support
WP III.6: Investigate how best to display counterexamples

Phase IV: Conducting case studies and evaluating our research
WP IV.1: Conduct two case studies provided by one industrial partner
WP IV.2: Conduct one case study provided by another industrial partner
WP IV.3: Draw conclusions and explore potential for commercial exploitation

Visits

Two one-week visits by Prof. Rance Cleaveland to York:
V1: To transfer knowledge regarding building simulators and to validate the semantics of Contractual Statecharts
V2: To discuss possible implementation avenues for model checking our refinement preorder

Three one-week visits to our project partners in Kiel, Germany (*V3–V5*)

Project meetings

Purpose of the project meetings to be held at York:
PM1: To kick start the project, bringing together researchers and industrial partner
PM2: To discuss the mixed-language approach with the industrial partner
PM3: To review the project at half time between investigators and research personnel
PM4: To progress the critical tool support phase together with our consultant Prof. Cleaveland
PM5: To conclusively evaluate the project and write the final report

Milestones

M1: Requirements for the Contractual Statecharts language captured
M2: Language definition, implementation and evaluation completed
M3: Refinement patterns and refinement checking completed
M4: Project completed and evaluated