# Exploring Applications of Formal Methods in the INSPEX Project

Joseph Razavi[1], Richard Banach[1], Olivier Debicki[2],
Nicolas Mareau[2], Suzanne Lesecq[2], Julie Foucault[2]

[1]School of Computer Science, University of Manchester,
Oxford Road, Manchester, M13 9PL, U.K.
{richard.banach,joseph.razavi}@manchester.ac.uk
[2]Commissariat à l'Énergie Atomique et aux Énergies Alternatives, MINATEC Campus,
17 Rue des Martyrs, F-38054 Grenoble Cedex, France
{olivier.debicki,nicolas.mareau,suzanne.lesecq,julie.foucault}@cea.fr

**Abstract.** As formal methods become increasingly practical, there is a need to explore their use in a variety of domains. Wearable sensing is a rapidly developing area in which formal methods can provide tangible benefits to end users, facilitating the advance of cutting-edge technology where consumer trust is critical. The INSPEX project aims to develop a miniaturized spatial exploration system incorporating multiple sensors and state of the art processing, initially focused on a navigation tool for visually impaired people. It is thus a useful test-case for formal methods in this domain. Applying formal methods in the INSPEX development process entailed adapting to realistic external pressures. The impact of these on the modelling process is described, attending in particular to the relationship between human and tool-supported reasoning.

## 1 Introduction

The industrial application of formal methods is becoming increasingly common. In safety-critical domains such as aerospace, train systems, and nuclear reactors, it is more and more the case that one can reasonably expect their use [1, 3, 7, 14]. For the design of CPUs, where the financial cost of failures is extreme, formal methods have become standard [22]. This is starting to extend to other types of widely used infrastructure such as operating system components and compilers [16], and famously, in the back-end operations of large web-based companies.[1]

These fields, of course, do not exhaust the range of potential applications. Indeed, a time may come when most software is developed using rigorous techniques, but this future is at present rather remote. Instead, the frontier consists of complex systems whose cost of failure is high, if not quite catastrophic. The development of systems with these characteristics presents an ideal opportunity for formal methods researchers and software engineers to engage with each other to make rigorous development more applicable and ubiquitous.

[1] We have in mind the use by Facebook [11] of behind-the-scenes verification tools, described in [32], and as predicted almost a decade earlier by Meyer in [20].

The area of medical devices is evidently one in which the consequences of errors may be permanently debilitating or fatal, and there, system construction is governed by numerous standards, e.g. [17] for software. Adjacent to life-critical devices, there is an expanding area of medical accessories, attempting to enhance the lives of their users in significant if non-critical ways. Among the many specific kinds of device in this category we mention 'assistive technologies' which aim to support users with specific needs to navigate a world principally designed without those needs in mind. If the device functions as it should, the benefit is an increased ability to live independently.

In this paper, we examine the INSPEX system [18], one example of a navigation aid for visually impaired people. Such navigation aids, if they are successful, may help the user to carry out more complex journeys than they would usually feel comfortable undertaking, and with less reliance on others to help them. From this benefit there arises a concomitant cost: if the system should fail, the user may be left stranded in an unfamiliar area which they would otherwise have avoided, perhaps even having to wait for assistance from friends, strangers or emergency services. While such an event would clearly undermine the increased independence which the device should bring, it is not the occurrence but the plausible probability of this kind of problem which is a threat to the system's usefulness: unless users can be reassured that failure is a remote possibility, they can not rely on the technology.

While the most classical navigation aid for visually impaired people, the white cane, is a simple and robust physical tool, the decreasing cost of sensors and increasing ubiquity of portable or wearable computing provides new possibilities to imagine assistive technologies. However, given the high reliability required and the complexity of the technology involved, the issue of correctness presents a barrier to entry for anyone wishing to provide such a product. Indeed, in some jurisdictions, devices of this type are highly regulated, underscoring the challenge to be met.

For these reasons, the development of assistive technologies represents a key area in which the industrial use of formal methods may expand. However, projects in this area are likely to be conducted under significant time pressure, and to be led to a great extent by technology and hardware development. Software components may be re-purposed from previous development efforts which are unlikely to have employed a rigorous methodology. These factors together eliminate two classical approaches for integrating formal methods into a project. A top-down approach becomes impractical, because the desire to leverage new technologies in a timely manner necessitates the use of existing components where possible. On the other hand, an incremental use of static techniques to analyse a system in deployment, as might be used for back-end technologies, is not practical for stand-alone devices which must be highly reliable from the outset.

In this paper, we describe the effects of these constraints on the formal modelling process, based on our experience working on the INSPEX project. In Section 2 the INSPEX project is described in more detail. Our experiences with different strategies for modelling under the constraints of the project are discussed in Section 3, and the extent to which we have been able to support this activity with existing tools is reported in Section 4. Concluding remarks are made in Section 5.

## 2 The INSPEX Project

The INSPEX project [18] aims to construct a wearable spatial exploration system, providing obstacle-detection and warning capabilities. Such systems in themselves are not new, and indeed the traditional white cane used by some visually impaired or blind people constitutes an example. Recently, advances in sensor technologies have made it possible for consumer applications to utilize advanced electronics for this purpose. This leads to enhanced white canes which incorporate range sensors such as ultrasound or LiDAR. While the cane sweeps to detect ground-level obstacles, the sensor can scan a head or body height, and the system can provide a warning beep or buzz if there is an obstacle in its path. A selection of existing or projected systems based around the advanced sensor idea includes Smartcane [28], Ultracane [31], Bawa [5] and Rango [25].

INSPEX will design a small, light device, suitable, in the first instance, to be mounted on a white cane to assist the blind and visually impaired. Further use cases include other low-visibility domains such as fire-fighting in smoke filled environments, or the operation of small airborne drones. INSPEX advances the state of the art for such systems in two ways.

First, incorporating ideas currently used for automotive applications, INSPEX combines readings from multiple sensors into a single statistical model of the environment [19, 27, 15, 30, 21]. Specifically, it makes use of a short range LiDAR, a long range LiDAR, an ultra wide-band RADAR, and a MEMS ultrasound sensor. This is a significant improvement over a single-sensor system because each sensor has different characteristics and each performs best under different circumstances. Factors such as light level, fog, rain, snow, reflectivity of the target or its distance and size, impact the accuracy of data from different sensing methods in different ways. Combining these diverse measurements can lead to greater accuracy, and discrepancies between them can reveal properties of environmental objects, such as translucency, which are not possible with one type of reading alone. As part of the INSPEX project, the sensors themselves have to be miniaturized and adapted to function in close proximity to each other. This work is carried out in parallel by the Swiss Center for Electronics and Microtechnology, the French Alternative Energies and Atomic Energy Commission, the Tyndall National Institute Cork, and SensL Technologies.

Second, the INSPEX device integrates a significant amount of processing, so that its output, rather than simple range readings, can consist of more meaningful data such as a depth-map of the scanned environment, or the location of salient obstacles. This saves the data consumer the effort of processing raw readings into a meaningful form. This is significant in human-oriented applications, as traditionally the presence of, or distance to, an obstacle in a particular direction is presented to the user rather directly, in the form of sound or tactile feedback, leaving the user's brain the task of extracting a model of the environment. This will be a familiar experience to those readers who have had to translate the more or less frequent beep of a car reversing sensor into sensible manoeuvres, especially in the presence of small unexpected obstacles. Relieving this cognitive load for users who must make constant use of the data from the sensors is significant, and the processing performed by the INSPEX sensing unit allows a smartphone application, developed with the French startup GoSense, to render the environment in a 3D ‘sound picture’, presented to the user via binaural headphones.

The system consists of many heterogeneous modules. There is the headset, the smartphone and the environment sensing system. Within the sensing system, which is where the focus of the technological development work is concentrated, the individual sensors are provided by autonomous submodules (capable of being deployed individually in other applications), whose readings are combined using a software processing subsystem for the digital information garnered.

These features make the INSPEX project an ideal test case for the application of formal methods to the class of problems described in the previous section. The success of the device crucially relies on its dependability, making formal methods attractive. However, time constraints, fundamental technological challenges, and the necessary re-use of existing components where possible mean that a pragmatic approach to the formal modelling and verification process must be taken. In what follows, we outline lessons learned from this project about how formal modelling can be incorporated in a development process with these characteristics.

## 3 Modelling Approaches for INSPEX

As outlined above, when modelling the class of systems of interest in this paper, one must frequently deal with bodies of existing code which have been modified so as to be suitable for the project at hand. In contrast to the ideal application of formal methods, in which formal modelling would be used to derive code from requirements, adding clarity and detail progressively, a more ‘bottom up’ approach is clearly needed which relates to the existing code as it is, and which acknowledges that requirements are somewhat obscure, encoded implicitly in implementation details and engineers’ minds.

In beginning such a modelling exercise, we have found that there are two tempting mistakes which must be avoided. The first is to take ‘bottom up’ too literally, and attempt to model the low-level of the code in complete detail. Under any plausible constraints on time and personnel, an unmodified interpretation of this is clearly impossible: in a system of any reasonable complexity, there are simply too many low-level events to lead to the extraction of a sensible formal model.

At the level where the functioning of the operating system and libraries is considered to be correct —perhaps leveraging existing work [13, 10]— the task has a semblance of achievability, but this quickly turns into a mirage. It rapidly becomes clear that while such a model may be constructible in the time available, there will be little time for anything else. Given that the constructed model would be almost a copy of the code as it is, little value would be added in terms of perception of the intended purpose of the system, at the cost of great effort. Discrepancies would be hard to detect because of the low-level focus, and, since the model would not be independent from the program, they would be vastly more likely to originate in modeller error than to be genuine defects in the system. While existing formalizations of operating system functionality are valuable tools for modellers of higher level systems, the insights contained in them must be more carefully deployed.

It must not be thought that these problems could be overcome by automated means. While models could in principle be extracted from code, ameliorating the issue of the quantity of work, the more fundamental problem of obtaining a copy of the existing low-level artefact would remain. In a system like INSPEX, it would be difficult to analyse such a model, even automatically, for problems more interesting than null pointer dereferences or buffer overruns. For example, the main correctness property we care about for the INSPEX system is liveness: we don't want the system to stop producing output unexpectedly or unreasonably. However, it is clear that there are lots of circumstances under which output is impossible.

In order to perform its analysis, the system requires sufficiently diverse input with sufficient frequency. What exactly is meant by 'sufficiently diverse' and 'with sufficient frequency' is not clear from a high-level point of view without considerable human insight. It crucially depends on implementation details, essentially corresponding to the way memory is managed. Even to understand that the problem can be stated in this form, reducing the number of properties of the implementation to be determined to just two frequencies, a high-level understanding of the code is needed. For the foreseeable future, extracting this kind of conceptual information from a computer program remains an unavoidably human task.

If the desire to start with the details as they exist is problematic, the opposite tendency is equally dangerous. Given that the salient aspects of the code are only visible in the light of a high-level understanding, it is tempting to try to run a traditional formal development, from requirements, through specifications, culminating in implementation-level models. These would then be compared with the existing code for discrepancies. Of course, the likelihood that low level models obtained in this way would match the real code would be remote. Furthermore, it would be a grave mistake to replace the real-world code with something generated by the model, since the real code embodies a wealth of practical experience about the efficient and robust implementation of the system which a modeller is unlikely to be able to replicate, particularly in little time. For this reason, the modeller must keep one eye on the code, though this starts to risk the same problem as the approach above: the specification is no good if it just amounts to saying that the abstract model does whatever the code does and thus the code is correct with respect to the abstract model by default!

A more formidable obstacle is the scale of the problem of going from low-level code to high-level properties suitable for a specification. As described above, coming to understand the precise requirements that the system places on its inputs in order to function properly depends on a detailed understanding of the code. It would be hubristic to imagine that a lengthy cogitation on the details would produce the required perspective for a system of any reasonable complexity.

Both options considered above share the defect that a lot of time and effort is consumed before any actual model results. This is a poor use of resources, as a widely drawn lesson in applied formal methods is that much of their value is derived from the human understanding of the system gained by producing models [14, 4]. These models (or from questions driven by constructing them) can be discussed with engineers, revealing points of tension which may imply the presence of inadequate understanding by any of the parties, or of bugs. This interactive process works best if comprehensible models can be produced early in the whole design and implementation activity as it is well known that the cost of fixing defects is roughly exponential in how far along the development route they are discovered [29, 8, 23].

The resolution of these dilemmas has two aspects. The first is that engineers will already have a conceptual understanding of the code they have produced, which is likely to be at an intermediate level of abstraction. They will be able to provide a description of the functioning of the system at the level of data structures rather than low-level manipulations. A model of this description has the advantage of being at the level engineers already think about the system, facilitating discussion and helping to resolve ambiguities in natural language descriptions. Data structures are likely to be motivated by non-functional considerations such as memory constraints or hardware requirements. In some sense, a description of the system at this level is likely to describe the *practical objectives* met by the code at a level which abstracts from detailed manipulations, but leave the *ultimate purpose* of the system's actions implicit. Therefore, in addition to making explicit how this description connects to the implementation details, the modeller must also extract a specification of correctness with a reasonable degree of independence.

Once this mid-level model is in place, the task of producing high and low level models is dramatically simplified. In a very idealized description, one might imagine working recursively, always attempting to make half-steps in the directions of specification and implementation simultaneously, and filling in the gaps between existing models. The distinctive property of this process of modelling is that refinement relations between models become formalizable all at once, as the end of the process of interpolation is approached. This delays the construction of a formal proof that the system behaves as it should, and prioritizes maximizing the amount of communication with the development team.

In reality, the situation is likely to be a little worse than what has just been stated. Producing models of the entire system, suitable to stand in relations of refinement to each other, becomes increasingly difficult as low-level features are incorporated and get in the way of clean abstraction. Instead, it is advisable to model whatever aspects of the system seem amenable to modelling.

For example, in the INSPEX system, the incoming sensor readings are pre-processed in various ways before being sent to the statistical algorithm which computes a representation of the user's environment. In the course of this processing, they spend time in various internal buffers. In a high-level approximation to the system, one imagines that the message contents themselves move around in these data structures, but in reality only references are manipulated. This generates some subtle requirements. When an abstract object simply disappears, its reference can not disappear: instead it must be used to deallocate the resource referenced. More interestingly, when a piece of abstract data passes from one buffer to another, in reality these buffers may be stored on separate subsystems. In that case, the reference must not be sent. Instead, the data itself must be sent, and the abstract value represented by a new reference to the copy.

In principle, these sorts of details are well captured by refinement, but in practice if the algorithm at an abstract level is already complex, a model incorporating the lower level details can become extremely unwieldy and in particular difficult to discuss with engineers. Instead, the processes implementing individual steps of the low-level memory management procedures can be modelled. The resulting set of models, then, will stand in a variety of relations to each other and to the code, focussing on select aspects of the system chosen by human judgement.

## 4 Tool Support

Most of the modelling work for INSPEX has been done using Event-B [2] and the Rodin tool [26]. The Event-B style of formal development constructs system models by building state machines, with state spaces (not by any means restricted to finite cardinalities) defined statically, and with the transitions between states defined by guarded events written in a guarded command language. The Rodin tool reasons about the consistency or otherwise of the model defined, by comparing the definition of the model's dynamics against the invariants and other properties that are included in the model's definition.

The choice of Event-B and Rodin was made principally on modelling grounds. We found that Rodin, together with its recently incorporated SAT solving plugins [9] doing the heavy lifting on the proving side, was very convenient for modelling timing-related properties of systems. It performed better for our application, and could be more useful, than tools which focus specifically on time. The reason is that systems such as ours do not fit well the perspective on time that those alternative tools take. By contrast, in many respects Rodin fits very well with the modelling process described here.

Of course, the ability to animate models using the ProB plug-in of Rodin [24] is very useful for communicating the meaning of models to non-specialists. In addition, the semi-interactive style of proof is well-suited to a modelling style in which much of the issue of correctness is left to human discretion. Indeed, we often find that arguments about data structures are often reducible by a combination of human and automated effort into an intuitively obvious statement. This can be marked as 'reviewed' in Rodin, allowing a record of the interplay of human and automated verification.

One might also think that some of the relationships between models alluded to above which do not amount to refinement may be covered by some of the many plug-ins available. Of the plug-ins available, those on model decomposition, and particularly atomicity decomposition [12], seem most likely to be relevant. At present, however, use of these tools presents the problem that one particular formalism out of many possibilities for decomposition must be chosen to represent a relationship between subsystems which is intuitively understood, but may correspond to each of the possibilities only imperfectly. This work only seems justifiable if it is reasonably clear that it would form part of a formal proof of correctness. We may return to this point in future work.

In addition to the relationships between different abstract models, there is also the question of the relationship between these models and the code. For detailed enough models, this relationship can be checked for plausibility by a human being, but this may lead to low-level problems being overlooked. In the INSPEX project, we have made use of the BLAST tool [6] to confirm aspects of our understanding of the code. BLAST was selected as an initial tool to investigate for this purpose because of the availability of tutorial material, and crucially because of its specification language which is conceptually close to the idea of guarded events.

For example, suppose that at a relatively high level of abstraction, we model a sensor process which first allocates slot from a buffer, then fills this slot with a reading from the sensor hardware, and goes back to waiting for free space to be available in the buffer. Schematically, a standard way to model a simple state machine of that kind would be to use a variable for the current state, an abstraction of the control state of the real program. To check that the real code corresponds to the model at this level of abstraction, one might write a BLAST specification in the following way. First, a new variable must be inserted into the code to model the state of the abstract system. To do this the BLAST specification might begin with `global int wait_for_buffer = 1;`. Next, events in the Event-B model are linked to the code by using BLAST events. In the Event-B model, the sensor being allocated space in the buffer would correspond to an event like the following.

```
GetBufferSlot
  WHEN  $state = wait\_for\_buffer$
        $available\_slots \geq 1$
  THEN  $state := wait\_for\_reading$
        $available\_slots := available\_slots - 1$
```

In the C code, the event might correspond to calling a function `allocateSlot()`, and the condition that there are available slots in the buffer might be indicated by a pointer, `next_slot` being non-null. Supposing that we are confident that the function `allocateSlot()` does reduce the number of free slots as required, and we only want to check that the control state machine is accurate, we might write a BLAST event as follows.

```
event {
   pattern { buffer_slot = allocateSlot();}
   guard   { wait_for_buffer == 1 &&
             next_slot != NULL }
   action  { wait_for_buffer  == 0; }
}
```

BLAST will check that whenever the ‘pattern’ in the above specification occurs, the ‘guard’ is true, and add the ‘action’ to the code to update the abstract state. This is somewhat like checking a refinement relation between the C code and the Event-B model. However, it relies on assumptions made by the modeller that the functions used behave as expected, and that there are no sources of control flow changes, such as failure to obtain a sensor reading or pre-emption by other threads, which have been ignored. Once these assumptions have been documented, they can be discussed with engineers, or used to guide the development of more detailed refinements of the Event-B models.

## 5 Conclusions

In this paper, we discussed the lessons learned from our formal modelling work on the INSPEX project about the way in which formal methods can be expanded into domains for which the usual accounts seem difficult to apply.

In particular, by treating formal tools as a way to explicitly represent human intuitions about the system, approximating the process of refinement by describing salient levels of abstraction, as much value can be drawn out of the modelling process as possible in limited time. The drawbacks of this approach are that the partial models constructed can stand in various relationships to each other, which may reduce the applicability of tools and delay the construction of proofs of correctness. In addition, describing the assumptions linking formal models with the real system can become complex. Nevertheless, formal methods can bring tangible benefits to projects where high reliability is important, but practical needs make a process structured around the use of formal methods unworkable.

**Acknowledgement:** This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 730953. The work was also supported in part by the Swiss Secretariat for Education, Research and Innovation (SERI) under Grant 16.0136 730953. We thank them for their support.

## References

1. Abrial, J.R.: Formal Methods in Industry: Achievements, Problems Future. In: Proc. ACM/IEEE ICSE 2006. pp. 761–768 (2006)
2. Abrial, J.R.: Modeling in Event-B: System and Software Engineering. CUP (2010)
3. Banach, R. (ed.): Special Issue on the State of the Art in Formal Methods, Journal of Universal Computer Science, vol. 13, (5) (2007)
4. Barnes, J.E.: Experiences in the industrial use of formal methods. Electronic Communications of the EASST 46 (2011)
5. Bawa: https://www.bawa.tech/
6. BLAST Tool: (2011), https://forge.ispras.ru/projects/blast/
7. Bowen, J., Hinchey, M.: Seven More Myths of Formal Methods. IEEE Software 12, 34–41 (1995)
8. Braude, E., Bernstein, M.: Software Engineering: Modern Approaches. Wiley (2011)
9. Déharbe, D., Fontaine, P., Guyot, Y., Voisin, L.: Smt solvers for rodin. In: International Conference on Abstract State Machines, Alloy, B, VDM, and Z. pp. 194–207. Springer (2012)
10. Divakaran, S., D'Souza, D., Kushwah, A., Sampath, P., Sridhar, N., Woodcock, J.: Refinement-Based Verification of the FreeRTOS Scheduler in VCC. In: Butler, Conchon, Zaidi (eds.) Proc. ICFEM-15. vol. 9407, pp. 170–186. Springer LNCS (2015)
11. Facebook:
https://en-gb.facebook.com
12. Fathabadi, A.S., Butler, M., Rezazadeh, A.: A systematic approach to atomicity decomposition in event-b. In: International Conference on Software Engineering and Formal Methods. pp. 78–93. Springer (2012)

13. FreeRTOS: (2017), https://www.freertos.org/
14. Hall, A.: Seven Myths of Formal Methods. IEEE Software 7, 11–19 (1990)
15. Hall, D.: Mathematical Techniques in Multisensor Data Fusion. Artech House (2004)
16. Harrison, J.: Formal Proof — Theory and Practice. Notices of the AMS 55, 1395–1406 (2008)
17. IEC 62304: https://webstore.iec.ch/publication/22794
18. INSPEX Homepage: (2017), http://www.inspex-ssi.eu/
19. Kedem, B., De Oliveira, V., Sverchkov, M.: Statistical Data Fusion. World Scientific (2017)
20. Meyer, B.: How You Will be Programming Ten Years From Now. In: ACM SAC-10 Keynote
21. Moravec, H. and Elfes, A.: High Resolution Maps from Wide Angle Sonar. In: Proc. IEEE ICRA (1985)
22. Pratt, V.: The Anatomy of the Pentium Bug. In: Proc. TAPSOFT-95. vol. 915, pp. 97–107. Springer, LNCS (1995)
23. Pressman, R.: Software Engineering: A Practitioner's Approach. McGraw Hill (2005)
24. ProB Tool: https://www3.hhu.de/stups/prob/
25. Rango: (2018), http://www.gosense.com/rango/
26. RODIN Tool: (2018), http://sourceforge.net/projects/rodin-b-sharp/ http://www.event-b.org/
27. Scalise, L., Primiani, V., Russo, P.: Experimental Investigation of Electromagnetic Obstacle Detection for Visually Impaired Users: A Comparison with Ultrasonic Sensing. IEEE Trans. on Inst. and Meas. 61, 3047–3057 (2012)
28. Smartcane: (2017), https://www.phoenixmedicalsystems.com/assistive-technology/smartcane/
29. Sommerville, I.: Software Engineering. Pearson (2015)
30. Thrun, S., Burgard, W., Fox, D.: Probabilistic Robotics. MIT Press (2005)
31. Ultracane: (2017), https://www.ultracane.com/
32. Verhoef, M.: From Documents to Models: Towards Digital Continuity. In: SAFECOMP/IMBSA-17 Keynote. https://drive.google.com/file/d/0B9DzO9PFER2xZDRxLUpKVUdYZmM/view?usp=sharing