SHARCS: Refinement-Centric Hazard Analysis of Requirements for Critical Systems

Asieh Salehi Fathabadi[0000-0002-0508-3066], Thai Son Hoang[0000-0003-4095-0732], and Michael Butler[0000-0003-4642-5373]

University of Southampton, U.K.
{a.salehi, t.s.hoang, m.j.butler}@soton.ac.uk

Abstract. The current state of art lacks a methodical, rigorous and scalable single approach to analysis of safety and security requirements. We present SHARCS (Systematic Hierarchical Analysis of Requirements for Critical Systems), addressing this gap by adopting an abstraction-based incremental approach to hazard analysis. SHARCS combines STPA-style control action analysis with Event-B formal modeling and refinement to analyze safety and security of cyber-physical systems by flowing down system-level requirements to component-level requirements. This paper summarizes our recent journal publication [6] with emphasis on two key contributions: (1) a systematic three-phase workflow (system level, component level, consolidation) that guides the incremental development process; (2) control abstraction diagrams, a novel visual notation for modeling control structures at different abstraction levels before concrete design. We demonstrate SHARCS on the Tokeneer secure enclave system, showing how these contributions enable scalable hierarchical decomposition with rigorous traceability.

1 Introduction

Safety and security are key issues in developing cyber-physical systems. Standards such as ED202A and ED203A [5, 4] mandate security risk assessments at different design stages.

Systems Theoretic Process Analysis (STPA) [11] and STPA-Sec [13] systematically identify hazards through control action analysis but rely on human judgement to assess effects. Event-B [2] provides rigorous formal verification but lacks systematic guidance for modeling choices. Previous work [3, 10] combined STPA with formal modeling, achieving both methodical analysis and rigorous verification, but lacks systematic support for incremental, hierarchical analysis needed for scalability.

This paper summarizes our journal publication [6] presenting SHARCS (Systematic Hierarchical Analysis of Requirements for Critical Systems), an abstraction-based incremental approach to hazard analysis. Our previous work [7] applied SHARCS to Tokeneer but did not formally specify the process. This paper advances beyond [7] with two contributions:

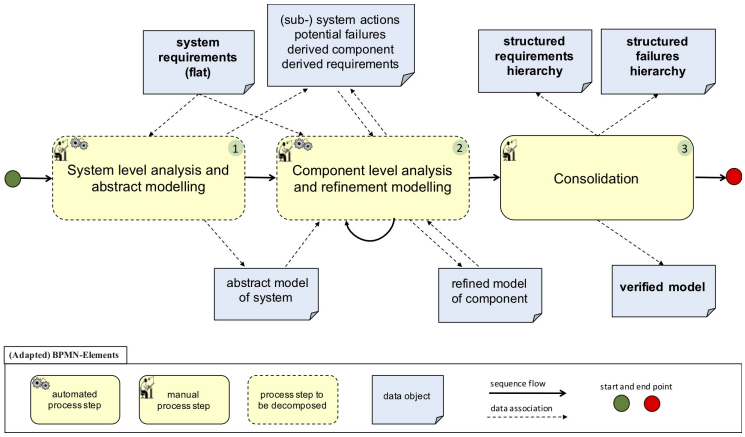

Fig. 1. SHARCS high level workflow

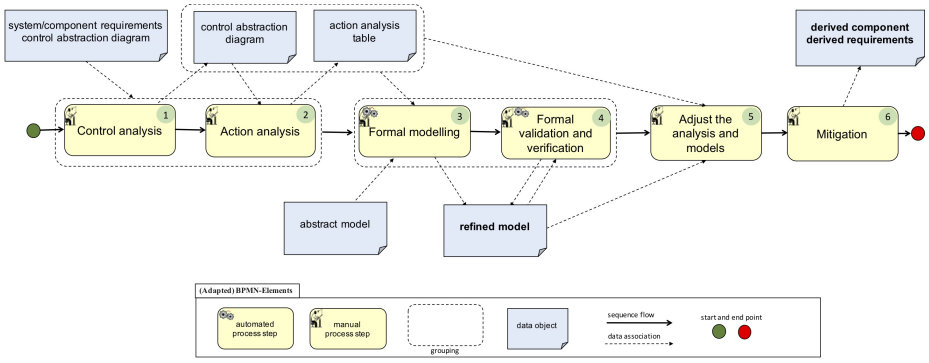

Fig. 2. Workflow for a single level of abstraction

1. **SHARCS Workflow**: A systematic three-phase process with detailed steps, inputs, outputs, and decision criteria (Section 2).
2. **Control Abstraction Diagrams**: Visual notation for modeling control structures at different abstraction levels before concrete design (Section 3).

We demonstrate these using Tokeneer (Section 3)[1].

2 SHARCS Workflow

Figure 1 illustrates the overall SHARCS process showing the main inputs and outputs of each level of analysis. The SHARCS approach consists of three phases: (1) system level analysis and abstract modelling, (2) component level analysis

[1] Full understanding of the technical details may require reading the journal publication [6].

and refinement modelling (repeated for each identified sub-system), and (3) consolidation phase.

System Level Analysis and Abstract Modelling: The main input to the system level analysis is a system requirements document. The requirements serve as the basis for construction of control abstraction diagrams and hence identification of system actions, potential failures, derived component requirements. These are used as input to the next component level analysis.

Each system-level consists of six steps, as shown in the expanded view of Figure 2:

Step 1, Control Analysis: The system requirements are used to construct a control abstraction diagram showing the actors involved and the information flow and control between them. This diagram identifies the control actions needed in the next step.

Step 2, Action Analysis: The control abstraction diagram is used to identify behavioural actions which are analysed to identify potentially insecure or unsafe actions. An action analysis table is constructed to analyse all the possible actions that can occur at the system level and identifies the resulting failures caused by that action occurring, not occurring or occurring with the wrong timing or order.

Step 3, Formal Modelling: The actions identified in the control abstraction diagram are then formalised in an Event-B model. Control actions are modelled by events that alter the state of the system and conditions are modelled as guards of these events.

Step 4, Formal Verification and Validation: The formal model is validated using model animation and scenario playing tools. When the model behaves in an appropriate way it is verified using theorem provers to ensure that the critical invariant properties of the system are maintained.

Step 5, Adjust the Analysis and Models: Design faults are corrected and the model is revised accordingly so that verification failures are eliminated. These improvements will involve strengthening the functionality allocated to component sub-systems, or introducing additional component sub-systems.

Step 6, Mitigation and Outline Design: Each insecure or unsafe control action is considered and addressed. Mitigating actions are proposed to prevent that action from leading to a failure. The outline design identifies the sub-system components and provides their broad derived requirements which are used as inputs to the next phase.

Component Level Analysis and Refinement Modelling: Back to the high level workflow (Figure 1), the component analysis phase is subsequently repeated if we identify further sub-components. The steps, Figure 2, involved in the component analysis phase are similar to those of the system-level analysis. The key differences are:

Step 1: Elaborate the control abstraction diagrams to add the control action structure of the component(s) introduced in this phase.

Step 2: Consider the component purpose, which has been identified as part of the previous level analysis and identify component failures. For certification

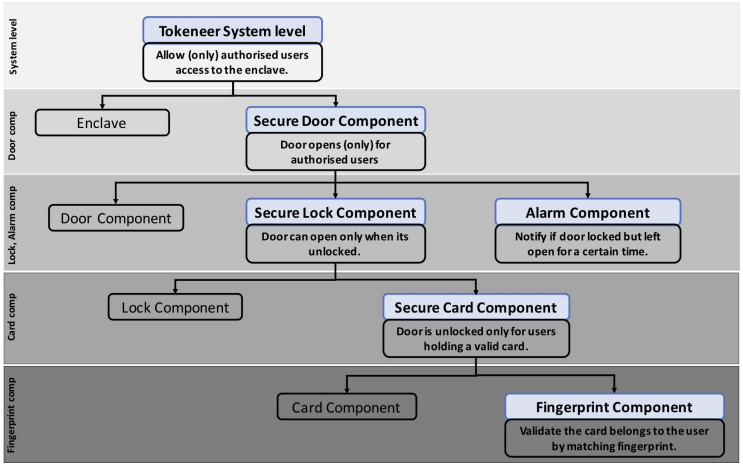

Fig. 3. Tokeneer: hierarchical component design and requirements flow-down

purposes, it is useful to record how the potential failures of this component link, via the control actions that this component addresses, to the previous level failures.

Step 3: Refine the abstract formal model to capture: component properties as invariants; refined/new events representing component level actions.

Step 4: Use automated theorem proving and model checking to verify constraints including the refinement proof obligations.

Consolidation Phase: In the consolidation phase (Figure 1) , we collate the results of the hierarchical analysis to produce an overview of the derived control structure and the verified derived requirements. This results in the hierarchical component structure (Figure 3 for Tokenner case study) and hierarchical failures. The final refined version of scenarios used in the hierarchical validation form an important output from the analysis as they illustrate scenarios that could potentially lead to a failure, and provide verification evidence that the design is robust enough to mitigate them.

3 Tokeneer Case Study and Control Abstraction Diagrams

We demonstrate SHARCS through the Tokeneer system, a secure enclave consisting of components including an ID Station that interfaces to a fingerprint reader, smartcard reader, door and visual display. The primary objective is to prevent unauthorised access to the Secure Enclave while allowing authorised users to enter and exit.

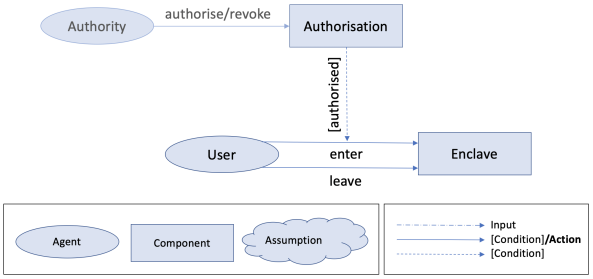

Fig. 4. Tokeneer system level control abstraction diagram

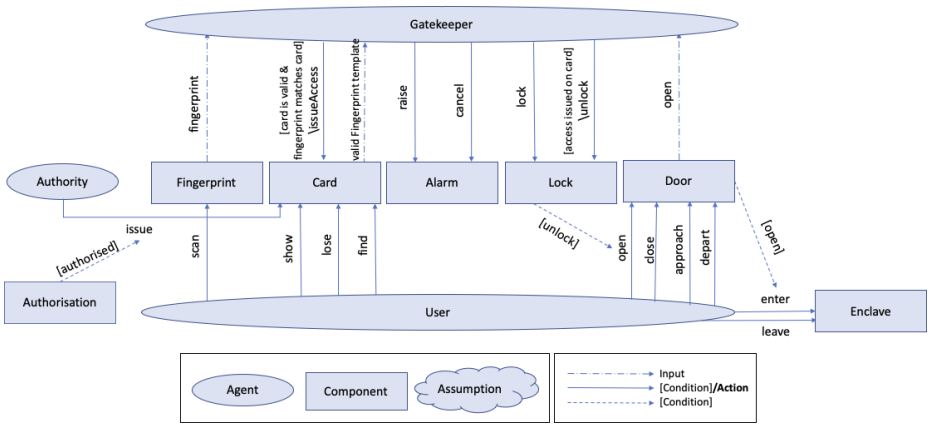

Fig. 5. Tokeneer fingerprint component level: complete control abstraction diagram

Control Abstraction Diagrams: Control abstraction diagrams, see Figure 4, visualize entities at a particular abstraction level with their information and control relationships. The diagrams show agents, components and assumptions as nodes and actions, conditions and input as links:
Agents (ovals) act on components via action links (solid arrows with labels). They may constrain actions via condition links (dashed arrows with [condition] labels).
Components (rectangles) are acted upon. They may be physical (enclave), conceptual (authorisation), or abstract (secure door). Abstract components are replaced by subsystems in future refinements.
Assumptions (cloud shapes) annotate behavioral properties the control relies on.
Inputs (dotted arrows) show information flow.

System Level, Abstract Control Structure: Figure 4 shows the Tokeneer system at the highest abstraction level. An agent, User, performs enter and leave actions on the physical component enclave. An Authorisation component

imposes constraint [authorised] on the enter action. An Authority agent can grant and revoke authorisation. At this level, *how* authorization is enforced is unspecified, addressed at the next level.

Action analysis examines these actions with respect to system failures. A system failure violates the system purpose, identified by negating it [2].

Complete System: Concrete Control Structure: The hierarchical component design leads to five control components over three refinement levels: secure door, secure lock, alarm, secure card and fingerprint (See Figure 3). Figure 5 shows the final control abstraction diagram. Components were added from right to left: door, lock, alarm, card, fingerprint. The gatekeeper agent was introduced with the lock component as the first with a physical control interface.

Considering Figure 4 which is an "abstraction" of Figure 5, containing 1 agents, 2 component, and 2 actions. Rather than performing hazard analysis across all the actions of the more complex Figure 5, we commence the analysis on the smaller number of actions of Figure 4. We then incrementally consider additional components and associated actions and analyse those additional actions for potential hazards.

Through refinements, see Figure 3, authorization checking shifts depending on component role: abstract at system level, enforced by Secure Door, flowing down through Lock and Card to Fingerprint which validates the card holder is the authorized user.

4 Related Work and Conclusion

STPA-Sec [13] and STPA-SafeSec [8] extend STPA for security but lack formal verification and hierarchical decomposition. Existing work combines STPA with Event-B [3, 10] or other formalisms [1, 9] but analyzes systems at a single abstraction level.

Our key contributions are: (1) **Systematic workflow** with clear phases, steps, and decision points; (2) **Control abstraction diagrams** enabling abstraction before design with direct Event-B mapping; (3) **Refinement strategy guidance**, one of the hardest tasks in Event-B is finding useful abstractions; SHARCS provides this by incrementally introducing components; (4) **Certification evidence** supporting ED-202A/ED-203A compliance [4, 5].

This paper summarizes our journal publication [6], emphasizing the SHARCS workflow and control abstraction diagrams that enable incremental decomposition from abstract system requirements to concrete component requirements. Application to Tokeneer demonstrates systematic identification of 5 component levels with formal verification across all levels[3].

Future work includes tool support for diagram-to-Event-B translation and parallel component analysis using Event-B decomposition [12].

[2] Full details are presented in [6].

[3] Available at `https://doi.org/10.5258/SOTON/D2957`

References

1. Abdulkhaleq, A., Wagner, S., Leveson, N.: A comprehensive safety engineering approach for software-intensive systems based on stpa. Procedia Engineering **128**, 2 – 11 (2015), proceedings of the 3rd European STAMP Workshop 5-6 October 2015, Amsterdam
2. Abrial, J.R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press (2010)
3. Colley, J., Butler, M.: A formal, systematic approach to STPA using Event-B refinement and proof. `https://eprints.soton.ac.uk/352155/` (2013), 21th Safety Critical System Symposium
4. Eurocae: ED-202A - airworthiness security process specification. `https://eshop.eurocae.net/eurocae-documents-and-reports/ed-202a/` (2014)
5. Eurocae: ED-203A - airworthiness security methods and considerations. `https://eshop.eurocae.net/eurocae-documents-and-reports/ed-203a/` (2018)
6. Fathabadi, A.S., Snook, C.F., Dghaym, D., Hoang, T.S., Alotaibi, F., Butler, M.J.: Systematic hierarchical analysis of requirements for critical systems. Innov. Syst. Softw. Eng. **21**(2), 569–593 (2025)
7. Fathabadi, S., Snook, C., Dghaym, D., Hoang, T.S., Alotaibi, F., Butler, M.: Designing Critical Systems Using Hierarchical STPA and Event-B. In: ABZ 2023: Rigorous State-Based Methods. Lecture Notes in Computer Science, vol. 14010. Springer (2023)
8. Friedberg, I., McLaughlin, K., Smith, P., Laverty, D.M., Sezer, S.: Stpa-safesec: Safety and security analysis for cyber-physical systems. J. Inf. Secur. Appl. **34**, 183–196 (2017)
9. Hata, A., Araki, K., Kusakabe, S., Omori, Y., Lin, H.: Using hazard analysis stamp/stpa in developing model-oriented formal specification toward reliable cloud service. In: 2015 International Conference on Platform Technology and Service. pp. 23–24 (2015)
10. Howard, G., Butler, M.J., Colley, J., Sassone, V.: A methodology for assuring the safety and security of critical infrastructure based on STPA and Event-B. Int. J. Crit. Comput. Based Syst. **9**(1/2), 56–75 (2019)
11. Leveson, N.G., Thomas, J.P.: STPA Handbook. Cambridge, MA USA (2018)
12. Silva, R., Pascal, C., Hoang, T.S., Butler, M.J.: Decomposition tool for Event-B. Softw. Pract. Exp. **41**(2), 199–208 (2011)
13. Young, W., Leveson, N.: Inside risks an integrated approach to safety and security based on systems theory: Applying a more powerful new safety methodology to security risks. Communications of the ACM **57**(2), 31–35 (2014)