Grand Challenges Vision Statement

Sam Procter, Software Engineering Institute

What types of systems will need to be developed in the future?

Heterogeneous Safety-Critical Systems As the use of software continues to increase in the systems we rely on for critical functionality – avionics, medicine, finance, communication, etc. – the components in those systems will become more specialized. Take for example sensors, which in previous generations might have been straightforward mechanical or electro-mechanical devices. In the short-term future (~5 years) they will increasingly have onboard computation capabilities to, for example: apply de-noising algorithms, encrypt output, or be remotely controllable. In the longer term (~10-20 years), given increasing computational power (bandwidth from technologies like 5g, more powerful CPUs, larger batteries, etc.) these trends will only accelerate.

Future systems will be composed of large numbers of these heterogeneous components from different manufacturers, who use different types of hardware, varying programming languages, and a variety of quality assurance processes. It will be difficult – or even impossible – for a system integrator to fully understand the complete behavior of each component. This will make component use, or even re-use, very challenging in high-assurance environments.

What "Grand Challenges" must be addressed to make these systems technically / economically feasible?

New Assurance Regimes Relying on Strong Notions of Component Interface The best hope we have of making assurance of these systems (e.g., safety, security, or airworthiness) tractable is to develop software and system engineering techniques that produce, and rely upon, strong notions of component interface. System components, particularly software components, could be more confidently used (and re-used) in system development tasks if we knew more accurately what the (side-)effects of using those components in a system would be. Safety literature, for example, is littered with stories of systems created using components which were developed for, and safely used in, previous systems. Once placed into new system contexts, though, fundamental assumptions that were made when those components were created were violated, and tragedies occurred.

Once created, these rich component specifications could be relied upon not only for evaluations of a component's functional role in a system, but also its role in the overall system's safety, security, and other critical characteristics.

What specific research areas / themes are associated with these challenges?

Interface Specification Languages Component interfaces should be specified in a language or set of languages that is flexible enough to describe the broad range of characteristics important to different stakeholders: both functional (e.g., behavioral

specifications) and non-functional (e.g., security characteristics, latency, safety) aspects must be easy to completely specify.

Tool Support for Interface Specifications Specifying those component interfaces must not be a burden: while auto-derivation of component specifications remains the “holy grail” end-goal, intermediate steps should be human-specifiable with minimal developer effort.

This task may seem impossible, but while it is quite broad (and may have to be focused / narrowed), computer science has tackled a number of very difficult problems before. This was done by first working out the theory and then implementing widely-supported tooling which is usable even by non-experts. Consider, for example, the creation of LR parsers or type safety / enforcement – early theory led to prototype tools which were refined gradually into software used by millions of developers around the world to eliminate entire classes of problems.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM20-1092