Massachusetts Institute of Technology
Lincoln Laboratory

BRASS Platform Evaluation
BBN - Interfaces, Models, and Monitoring for Resource-aware Transformations that Augment the Lifecycle of Systems (IMMoRTALS)

George Baah, Timothy Braje, Karishma Chadha, Jeffrey Diewald, Jeffrey Hughes, Mike Jones, Albert Reuther, Siddharth Samsi, Cassandra Sparks, Konrad Vesey

January 11, 2017

This work is sponsored by Defense Advanced Research Projects Agency (DARPA) under Air Force contract FA8702-15-D-0001. Opinions, interpretations, conclusions and recommendations are those of the author and are not necessarily endorsed by the United States Government.

Lexington Massachusetts

This page intentionally left blank.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

LIST OF FIGURES

This page intentionally left blank.

LIST OF TABLES

This page intentionally left blank.

1. INTRODUCTION

1.1 PLATFORM OVERVIEW

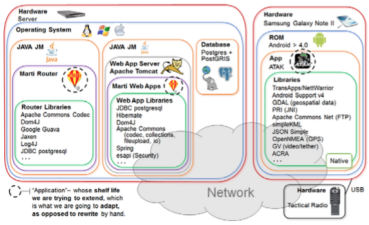
[AR – copied from BBN proposal: V. Technical Plan section]

A widely used tactical Situational Awareness (SA) Information Management (IM) platform is the basis of the IMMoRTALS approach to resource aware adaptation of software systems. The platform consists of Android-based clients and server-side applications with which they interact. A typical application running in this platform is shown in Figure **??**. The Android Team Awareness Toolkit (ATAK) application enables deployed units to keep track of each other, share threat or disaster information, and collaboratively handle threats or perform recovery operations. The ATAK application is served by a publish-subscribe based message router, a database for persist- ing SA data and imagery, and a web frontend for the database. Figure **??** also shows the systems resources and the software ecosystem of this platform. Compute, storage and communication resources change not only as Android devices and server hardware and wireless networks evolve, but also based on mission needs. For instance, in a forward deployed operation, the server may need to be hosted on a flying asset that can only support hardware equipment such as a Raspberry PI, or the area of operation may require using a low-bandwidth point-to- point ad-hoc network as opposed to a high bandwidth 4G cellular service. BBN and Vanderbilt researchers have extensive experience with this platform and have manually adapted SA applica- tions, essentially software product lines, to accommodate various changes.

The IMMoRTALS approach to discovery (of resource dependencies) and analytics (to detect salient changes and incorporating the transformations required to accommodate them) is in- formed by the richness of the platform and the teams insight about the change drives encoun- tered by this platform. OSU, Syracuse, and Securboration provide key discovery and analytics technology componentry and expertise that complement the capabilities that BBN and Vander- bilt bring to the team. The discovery concept of operation is shown in Figure 4. Building upon the concepts of product line architectures (PLA), IMMoRTALS discovery services will introduce a DSL to express inferred as well as desired resource requirements of PLA components, where the term resource is used in a generalized sense to mean systems resources such as compute, storage and communication resources, as well as the software ecosystem that the application needs to compile and run. The PLA components are defined in terms of Discrete Functional Units (DFUs), which are program units such as classes and functions that map to a Feature Model (FM) of the SPL. The inferred resource requirements, resulting from program analysis, form the extra functional specifications (EFS) of existing components. The desired resource requirements, re- sulting from the analysis of detected salient changes in the systems re- sources and the ecosystem and the EFS of the current configuration, drive the selection and code trans- formation as part of the IMMoRTALS analytics process.

1.2 PLATFORM FOR TEST

[AR – copied from BBN proposal: V.3 Platform, Challenge Problems and DAS section]



(a) System (b) Software Architecture

Figure 1. Platform software architecture.

Our platform manages tactical situational awareness (SA). SA is a particularly elastic problem: the more SA you have the better, and graceful degradation can be accomplished with less frequent updates or only sending updates when there is a change people would care about. It is easy to enhance or expand the problem by adding new sensors or more people.

A typical use of this platform comprises Android devices, acting as SA sensors or SA consumers or both, a pub-sub router which uses a database for storing large SA objects, and a web service to expose the stored data on demand (e.g., a tactical client gets a thumbnail with a URL, and the user can choose to click on the link at an opportune time).

Let us consider a basic SA application that runs on this platform. The client part of the application runs on Android devices, and uses GPS to report the users position to the network, and plots both self-position and the positions of other devices on a map. The server part of the appli- cation runs on a radio-connected Linux host, and acts as the publish-service broker routing SA messages between the Android clients. The client application was originally designed for use in mobile ad-hoc networks (MANETs) and makes use of IP multicast. We later evolved it to be used in a wide variety of other networking contexts, from various military tactical radios to commercial cellular (3G/4G). As the application evolved to use different networks, the use-cases evolved as well. Whereas before it was used to coordinate small groups (¡20) with line-of- sight tactical radios, now it is used to coordinate multi-agency operations over long distances.

For another representative application for this tactical information management platform, con- sider a face recognition program used to spot suspects in a crowd or a deployment area. Face recognition algorithms such as Linear Discriminant Analysis (LDA) and Bayesian Analysis offer different accuracy, but also consume different resources. Image format and ability to use wide angle

and zoom also impact image quality, the size of data to be analyzed, and detection accuracy. As deployed devices start to offer sophisticated cameras and additional CPU and memory, the initial application also needs to adapt, e.g., changing the algorithms and location to run them.

We will base our CPs on tactical SA applications like these-both client and server side components, and including pre-mission as well as in mission change drivers.

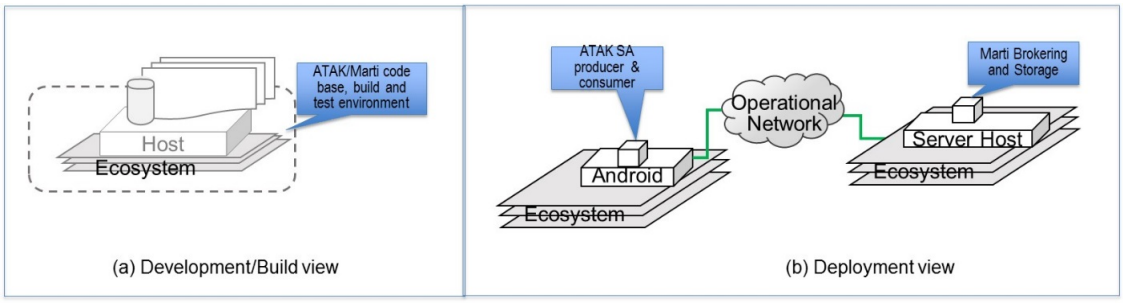


Figure 2. Test Platform

[AR – copied from MIT wiki]

Tactical situation awareness is the domain chosen by IMMoRTALS. The IMMoRTALS platform is a collection of hardware and software that supports domain operation, e.g., a tactical SA missions. The IMMoRTALS platform consists of Android based clients and a Marti server that facilitates interaction among the Android clients. The client produces a self-location message periodically, and subscribes to position reports sent by other clients. The servers most basic functionality takes position reports and forwards them to all connected clients. Variations on the client and server include sending other data besides position reports, such as images or other sensor data.As such, the development/build view of the baseline (i.e., without any IMMoRTALS technology) platform consists of the client and server code base, and build and test tools (Figure 2(a)), and the runtime view of the baseline consists of the Android client(s) connecting to the Marti server over a network (Figure 2(b)).

The ecosystem includes "resources" that the SA applications the Android client app and the Linux server side application- rely on to function, including both SW and HW resources. That is the ecosystem includes systems resources such as CPU/memory/network, software elements such as libraries and packages, as well as other services that must be running.

The IMMoRTALS DAS is the baseline platform + all our technologies to detect and respond to changes + an interface/instrumentation to inject change and observe our response (see Figure 3).

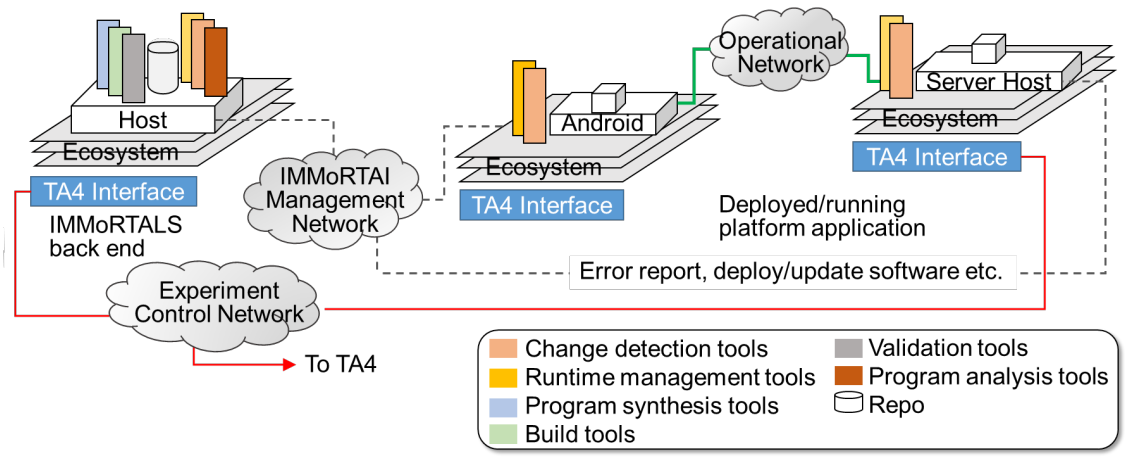


Figure 3. DAS Topology

As the legends in Figure 3 shows, IMMoRTALS technology, in the form of software tools and services, will be used in both the development platform and the deployment platform to support effective and efficient resource-aware program adaptation.

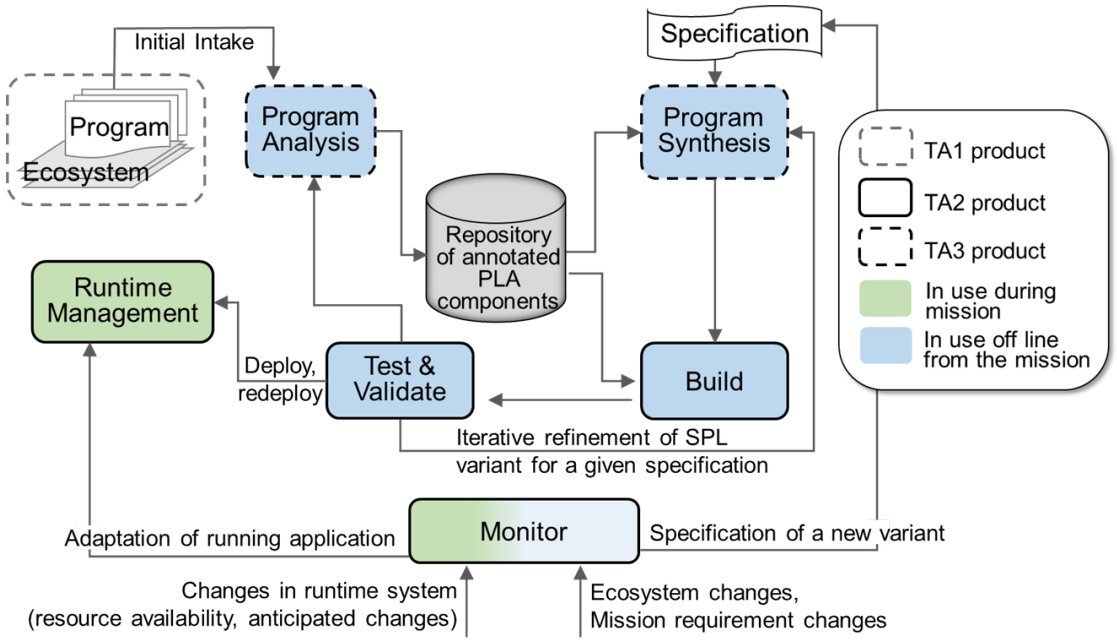


Figure 4. Immortals architecture

Figure 4 shows the end to end operation of the IMMoRTALS DAS covering both the development platform and the deployment platform. “Offline” changese.g., changes in mission requirement, a new library update are mostly handled in the development side. Changes in systems systems resources can be both ”online”/runtime as well as “offlne”. For example, CPU, memory or network resources may diminish in the deployment environment due to failures, load, attacks, contention with other applications etc. At the same time, the build platform can be used to adapt an existing version of the application for a new deployment environment with different systems resource profile.

The “repository” element is a logical entity, but our current implementation plan is that a SPARQL server will be at the core.

Our intent is to have the end-to-end “offline” workflow in place for the initial evaluation.

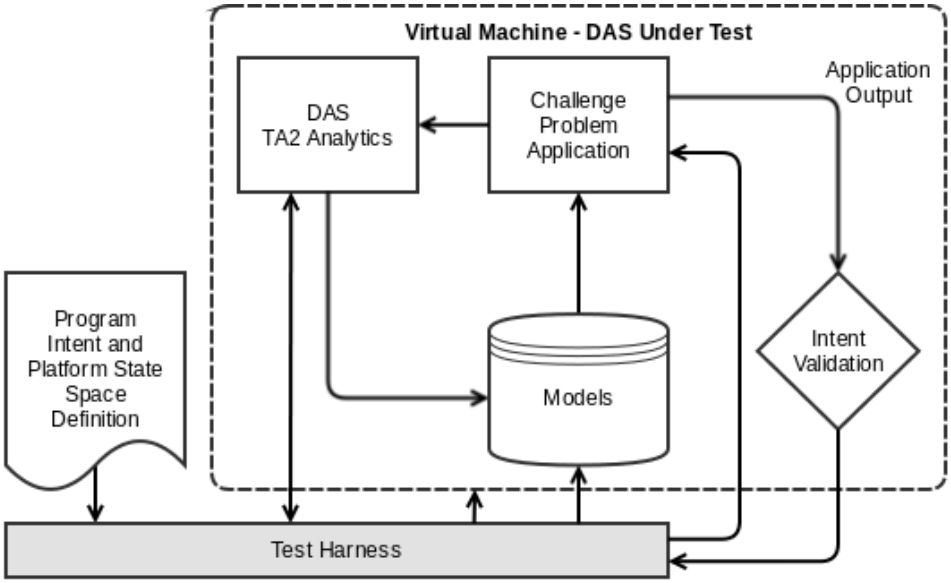


Figure 5. Notional challenge problem evaluation diagram

IMMoRTALS DAS generally fits into the structure depicted in Figure5 – except that it may not be a single VM.

The runtime components of the IMMoRTALS platform (i.e., the Android client and the Marti server) will be on their own VM. A separate VM (Linux Router?) may be needed to emulate the operational network and offer the TA4 interface to effect changes.

The models, analytics and the discovery machinery will be in the build environment/development platform. Some runtime change detection and runtime managers may be instrumented into the runtime VMs (i.e., the VMs hosting the Android clients and the Marti server).

This page intentionally left blank.

2. EVALUATION METHODOLOGY

2.1 EVALUATION TERMINOLOGY

[Note – This section defines a common set of terms used throughout this document.]

2.2 EVALUATION CRITERIA

[Note – This section describes a general approach for evaluating all of the different BRASS platforms.]

2.2.1 Intent Discovery

2.2.2 Analysis and Detection

2.2.3 Adaptation

This page intentionally left blank.

3. CHALLENGE PROBLEM 1: MISSION REQUIREMENTS REGARDING LOCATION COLLECTION

3.1 OVERVIEW

[AR – copied from MIT wiki]

Tactical situational awareness spans a large variety of missions. Such missions might have very different requirements for location collection. Some deployments may be satisfied with the GPS built-in to a given Android device. Other deployments might require GPS providers of a very specific nature (e.g. SAASM). Still other deployments might have to assume a GPS-denied environment, where the only options are dead-reckoning and potentially inertial sensors.

The accuracy and trustworthiness of the source of a location report come into play in a variety of ways from a mission requirements perspective. For instance, some use-cases by get by with only lat/lon values. Other mechanisms, such as dead-reckoning, can be augmented with a database of elevation data (DTED), at the cost of significant local disk space. Some of the potential mission requirements imply specific hardware being available. Even then, there can be multiple ways to get access to that hardware, for instance wired/USB vs. Bluetooth for accessing an external GPS.

This is primarily an "offline" adaptation scenario in its initial incarnation, although later iterations of this CP could easily be extended to have some online aspects. For instance, a mission requirement could be that the application use GPS when available, but be able to dead-reckon if there is no GPS. A further evolution might specify all possible location sources be built-in, and a preference-ordering is supplied as a mission requirement.

This CP can be used to validate the system in a variety of ways. Consider scenarios where there is no code in our repository that can satisfy the requirements versus scenarios where such code does exist:

Let us assume that the baseline version of the product is built to use GPS location.

- (offline) A mission requirement to use a different location provider comes in, IMMoRTALS finds a location provider that satisfies the mission need in the existing ecosystem
- (offline) A mission requirement to use a different location comes in, but there is no location provider that satisfies the mission need – may be the semantics do not match, maybe the resource profile of the available provider does not match with what is available in the deployment target
- *PHASE II / FUTURE* (online) GPS signals became unavailable during mission, but the deployed program or the deployment environment includes an alternate provider and the system adapts to use the alternate provider
- *PHASE II / FUTURE* (online) GPS signal became unavailable during mission, but the deployment set up did not include any alternatives to fall back on, a failure report is provided to IMMoRTALS

– At this point, this could be treated as the beginning of an "offline" change handling scenario

Note that adapting to offline changes may not be fully automated, IMMoRTALS will likely provide a set of change directives that a human programmer will have to encode. In later phases we will look into automating this step using program rewriting.

3.2 TEST SPECIFICATION

3.2.1 Overview

The Test Procedure documents all test relevant information necessary to reproduce the tests supporting the evaluation results and discussion for this challenge problem.

Mission

[AR – copied from MIT wiki]

There are two system perturbations that this challenge problem exposes. First is the mission requirement for a given source of location data. Second is the execution environment, which will dictate availability of specific hardware or external services (e.g. GPS satellites).

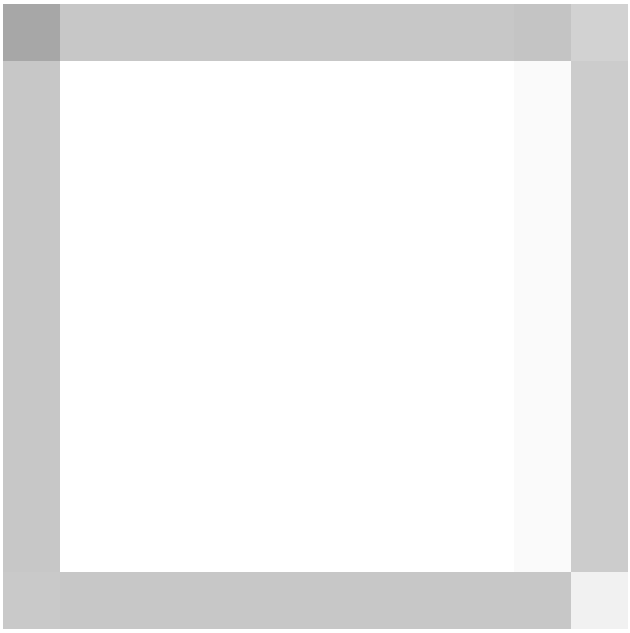


Figure 6. CP1 mission scenerio.

We have a set of possible mission requirement-based location collection techniques such as

- Dead-reckoning
- Inertial tracking
- Consumer-grade GPS
  - Built-in
  - Bluetooth
  - USB
- SAASM GPS
- *PHASE II / FUTURE:* DTED (elevation only; assumes another mechanism to find lat/lon)

A given deployment might specify that some options are required. We anticipate this will be input to the DAS as a checkbox or property file (see evaluation API section below).

Platform for Test [MIT LL – Missing description of platform for CP1] **Capabilities of the Baseline System** [MIT LL – Missing high level description of CP1 baseline system functionality]

General Interaction Model [MIT LL – Missing description of interaction between TA4 and CP1 system (and figure)]

Figure 7. Overview of CP1 interactions with the MIT LL test harness.

TABLE 1

CP1 Perturbation, Detection and Adaptation Timeline

Apply Perturbation	Offline
Detect Perturbation	Offline
DAS Adaptation	Offline

Evaluation

[AR – copied from MIT wiki] [MIT LL – Missing verdict expression for evaluating test results]

The set of possible execution environment properties might include:

- GPS satellite availability
- GPS chipset type (e.g. SAASM)
- External GPS connectivity (wired/Bluetooth)
- *PHASE II / FUTURE* Availability of sensors appropriate for inertial tracking

These could be set over the TA4 interface the IMMoTALS DAS will provide.

Figure 8. Overview of CP1 evaluation.

[AR – copied from MIT wiki]

The code that the DAS has available in the repository will have an effect on the results. Submitting a mission requirement for a SAASM GPS might return a working application or declare

it infeasible, depending on whether there is code for collecting SAASM GPS data in the repository. So the contents of the DAS code repository might be another system perturbation.

3.2.2 Virtual Machine Specification

The Virtual Machine (VM) Specification outlines the baseline software and hardware relevant environment as delivered by the platform team as well as that used during evaluation.

TABLE 2

Virtual Machine Specification

	Delivered	**Evaluated**
Processor	single core x86_64	TBD cores x86_64
Memory	4 GiB	same
Disk	30 GiB	same
Operating System	Ubuntu 16.04 64-bit	same
Other Software Requirements	Android SDK and build environment; Java JDK 1.8	same

3.2.3 Test Harness API

This section describes the platform and challenge problem specific *Test Action Messages* as defined in the BRASS Phase 1 Communications API document [1]. Further communication channels supporting *Test Control Messages* for test orchistration, *Test Data File*, and *DAS Log File* for test configuration and logging are to be implemented exactly as described in the BRASS Phase 1 Communications API document and will not be detailed here.

[AR – copied from MIT wiki]

The mechanisms for perturbing the system for this problem consist of constructing a new set of mission requirements or execution environment descriptions. We will provide a set of properties and their possible values in each of those categories. We will also provide a pre-populated repository of code fragments.

Having actual SAASM hardware as part of the evaluation is infeasible, as well as strictly controlling GPS availability. So we will provide drivers that simulate some of the notional code fragments described above. An evaluator will be able to tell both by looking at the code for the final application (e.g. the application instantiates SaasmGpsLocationProvider), and by some markers we will put in the output data, which location provider was chosen.

[AR – copied from Interface API section of MIT wiki post]

Our DAS implements a single REST call that takes a parameter in the form of a TTL (Terse RDF Triple Language) file. The TTL file describes the mission requirements and desired deployment options, including descriptions of the deployment hardware, software, and resource constraints, such as bandwidth or the availability of external services. The TTL file contains much more detail than is required for our Phase 1 Challenge Problems, so we intend to provide a script that will allow easy tweaking of the parts we are addressing in Challenge Problems 1 and 2, while filling in the rest of the model from a template.

Between CP1 and CP2 we have a total of 9 possible dimensions of change. Some of these dimensions are a simple binary choice (such as whether the target Android device has bluetooth capability), and some are numerical ranges (e.g. positive integers for number of clients). The script and resulting TTL file will contain all the settings from both CP1 and CP2, so both challenge problems can be evaluated with the same infrastructure. If there is a desire to evaluate CP1 and CP2 independently, we will provide some nominal settings when evaluating CP1 to use for the CP2-specific dimensions and vice-versa (e.g. the dimensions that only apply to CP2 don't change when making perturbations for CP1).

Once a TTL file is obtained, whether using our script or another method, an HTTP POST call is made to our DAS with that file as a parameter. When that HTTP POST call returns, assuming it returns a SUCCESS code, new application binaries should be available for deployment and scenario execution.

[AR – there is not enough information to populate the API yet]

[Note – API to be updated.]

3.2.4 Parameter Specification

Listings 1 documents and describes/defines the parameters used in this challenge problem to perturb the ecosystem in which the DAS will be evaluated. Each parameter is defined in the format described in the MIT LL ParameterDSL document [2] and captures 1) parameter name 2) baseline value 3) constraints to be satisfied when choosing new values and 4) comments providing more information about each parameter.

The following definition provides guidance on the structured format for defining the parameters using the MIT LL ParamDSL.

[AR – The parameters were copied directly from the Parameter Specification section of the MIT wiki post.]

The baseline values defined for each parameter alway represent the conditions used when evaluating the system as delivered in its baseline state.

```
// Comment
OptionalNamespace : {
    IdentifierName1 : baseline_value1 , constraint1;
    IdentifierName2 : baseline_value2 , constraint2;
    ...;
};
```

Listing 1: Challenge Problem 1 Paramters

```
// Mission Requirements
trustedLocations: True, no_constraint;

// Location Providers and Interfaces
gpsSatellites : True, no_constraint;
bluetooth: True, no_constraint;
usb: True, no_constraint;
internalGps: True, no_constraint;
userInterface: True, no_constraint;
```

3.2.5 Test Procedure

[Note – This section should define and describe the goals of each test case(s), trial, test suite, etc. necessary to evaluate the platform.]

3.2.6 Automata

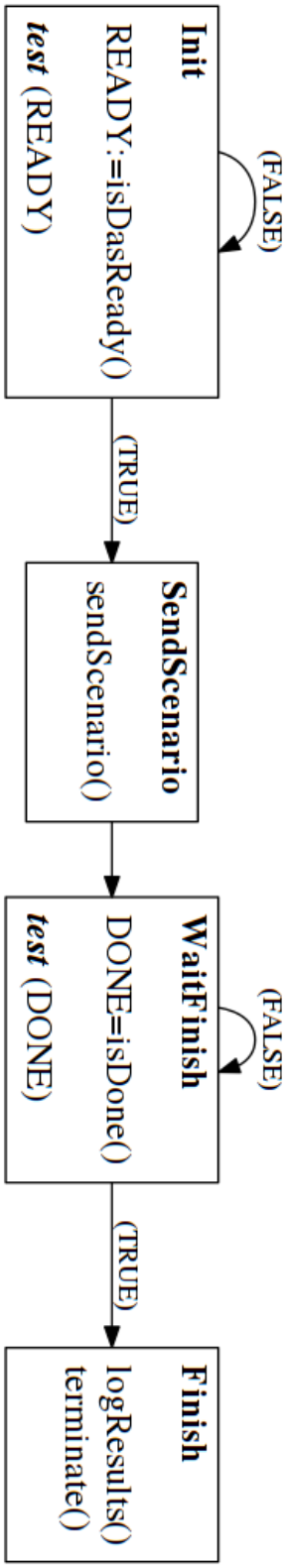


Figure 9. CP1 Automata.

TABLE 3

Functions used in the CP1 Automata

isDasReady() :=
    GET /DASSTATUS == Operational

startTest() :=
    POST /start_challenge_problem

3.2.7 Intent Specification and Evaluation Metrics

Element1 [MIT LL – Missing intent element description.]

TABLE 4

CP1 Intent Element 1: Element1

Intent: Element1	
Informal Description: Description.	
Formal Description: Description.	
Test Measurement: Test.	
Result Expression: (a, b)	
Verdict Expression:	
Mission Evaluation	
a ≤ b	Complete
a > b	Not Complete
???	Inconclusive
???	Invalid
???	Error
Quantitative Evaluation (Baseline vs. SUT)	
condition1	Enhanced Capability
condition2	Similar Capability
condition3	Degraded Capability
Mission Evaluation ≠ Complete	Inconclusive

3.3 EMPIRICAL EVALUATION

[Note – This section to be populated with key results by the Platform Workshop. More description and refinements to be made after the Platform Workshop.]

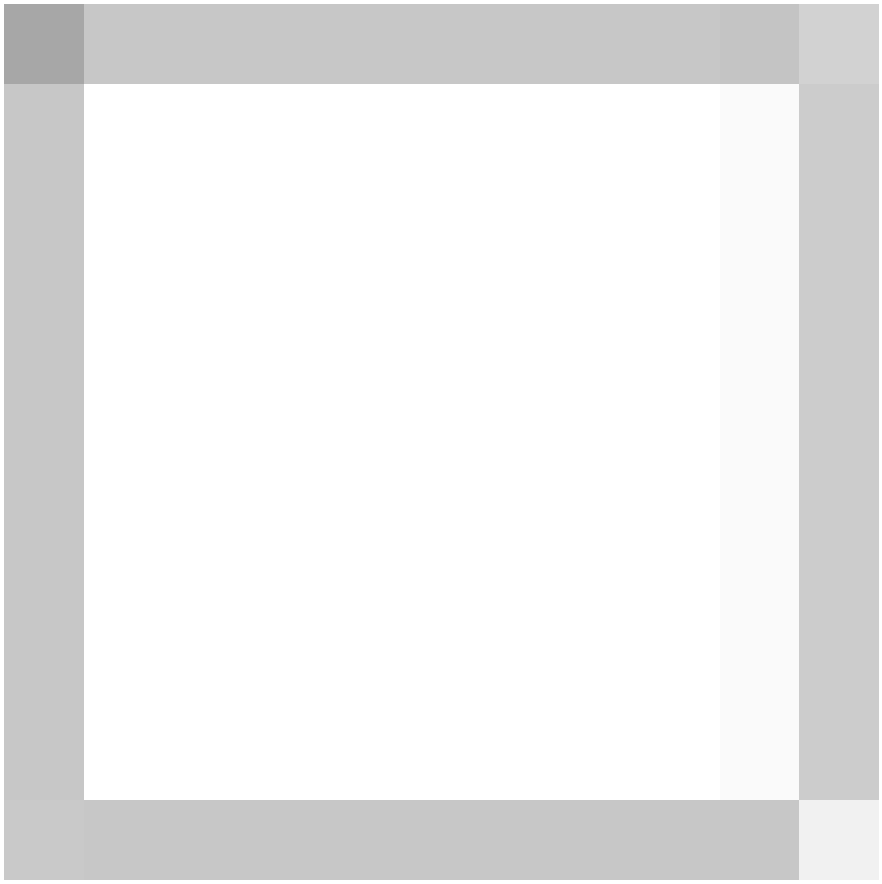


Figure 10. CP1 Summary Result

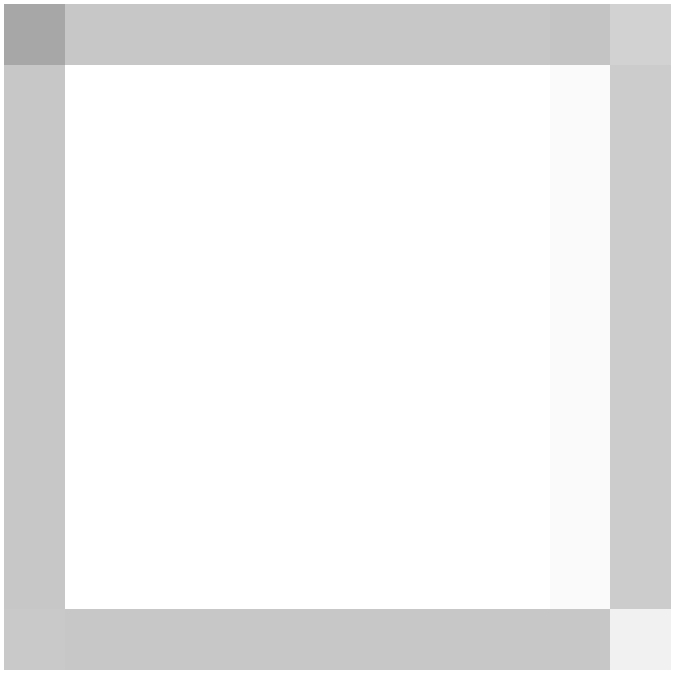


Figure 11. CP1 Key Result 1

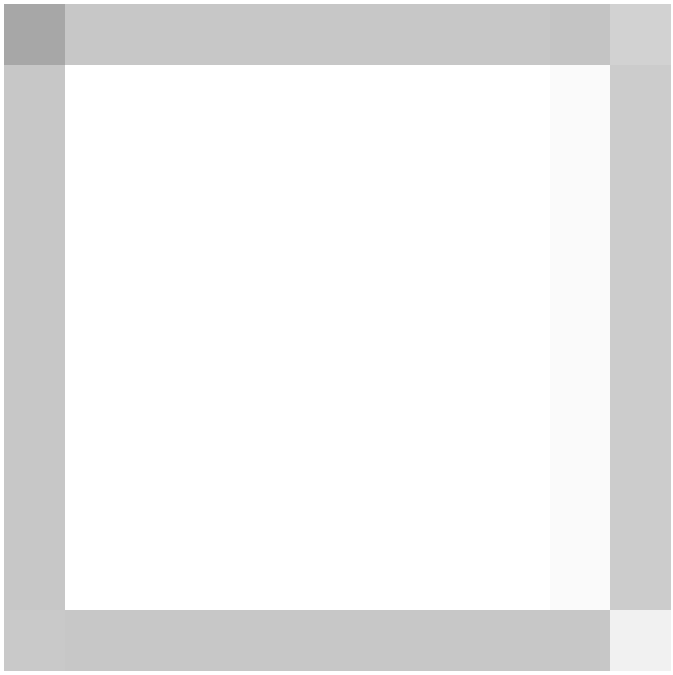


Figure 12. CP1 Key Result 2

3.4 DISCUSSION AND RECOMMENDATIONS

[Note – This section to be completed after the Platform Workshop.]

3.5 OVERALL ASSESSMENT

[Note – Discuss strengths, weaknesses and suggestions for improving this challenge problem]

3.5.1 Intent Discovery

Correctness

Completeness

Specificity

3.5.2 Analysis and Detection

Sensativity

Stability

Performance

3.5.3 Adaptation

Fidelity

Target System Performance

DAS Performance

3.5.4 OPTIONAL: Platform Specific Criteria

Expressivness of Specification Language

DAS Learning

Software Development and Maintenance

Impact Analysis

System Resilience

This page intentionally left blank.

4. CHALLENGE PROBLEM 2: HANDLING THE IMPACT OF CHANGES IN DEPLOYMENT SCALE AND OTHER FACTORS THAT IMPACT RESOURCE CONSUMPTION

4.1 OVERVIEW

[AR – copied from MIT wiki]

Tactical SA can vary in scope from small-squad (<10 participants) missions to major National Security Events (>500 participants). Along with the number of participants, other variables might refer to expectations of the mission, such as report rate or additional data beyond basic Position Location Information (PLI) (e.g. pictures). All these factors have an impact on the resource usage of every node in the system, but in most cases the limiting factor is bandwidth. In many cases the cause for limited bandwidth is purely technological, but in other cases bandwidth may have outsized cost (e.g. SATCOM) such that users want to voluntarily limit usage even though the technology would support more bandwidth.

The goal of this challenge problem is to see whether IMMoRTALS can evolve the client and server applications in response to such variations. Like CP1, this CP is also a family of challenge problems. We outline a number of possibilities, and indicate what we propose to achieve in Phase 1.

- The change request comes in, the code to use exist in the current ecosystem or the current system is amenable to changing/tuning the contributing parameter
  - Obvious boundary cases: the deployment environment can only accommodate a certain scale. For example, for a given bandwidth limit, it is easy to set number of client, rate and size in such a way that no solution can existIMMoRTALS should indicate these impossibilities
- Same as above except the code to use is not available, and there is no way to tune the offending parameters
  - The goal here is to eventually generate new code that fits and adapt existing code to allow parameter tuning. Initially, IMMoRTALS will just prescribe the changes. We will try automated handling of this case in later program phases
- *PHASE II / FUTURE* Online: rate or number of clients changed during mission and the deployed system is able to accommodate the changes
  - Same caveat about boundary casesit will be easy to set parameters that the deployment environment cannot sustain

4.2 TEST SPECIFICATION

4.2.1 Overview

The Test Procedure documents all test relevant information necessary to reproduce the tests supporting the evaluation results and discussion for this challenge problem.

Mission

[AR – copied from MIT wiki]

There are several system perturbations that this challenge problem exposes. The resource constraint that is most prominent is bandwidth. As mentioned in the overview, a bandwidth limit might be hard or soft.

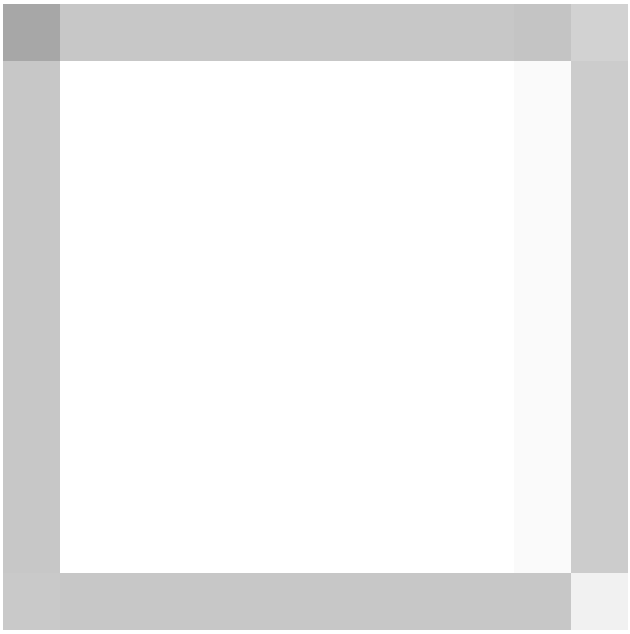


Figure 13. CP2 mission scenerio.

[AR – copied from MIT wiki]

We envision a set of possible mission requirement variables:

- Number of clients [MIT LL – Missing brief description of each of these requirement variables]
- PLI Report Rate
- Image Report Rate

- *STRETCH GOAL for PHASE I:* Image property requirements (e.g. resolution, color vs. grayscale)

Some deployments may not have a requirement for images. The PLI report rate and number of clients will be in every scenario description

Platform for Test

[AR – copied from MIT wiki] [MIT LL – IMMoRTALS team to review and refine]

The most critical aspect of the execution environment for this CP is bandwidth availability. For the first iteration, we will be considering a simplified network model where we only consider the "most constrained link". This is sufficient to describe most deployments, especially where the actual network topology may be opaque (e.g. commercial cellular). Later iteration may expand on the network definition to allow more complex topology descriptions.

[AR – copied from MIT wiki]

As in the first CP, availability of code in our repository to satisfy certain functional requirements is something that could be adjusted to get different results from our system.

Capabilities of the Baseline System [MIT LL – Missing high level description of CP2 baseline system functionality]

General Interaction Model [MIT LL – Missing description of interaction between TA4 and CP2 system (and figure)]

Figure 14. Overview of CP2 interactions with the MIT LL test harness.

TABLE 5

CP2 Perturbation, Detection and Adaptation Timeline

Apply Perturbation	Offline
Detect Perturbation	Offline
DAS Adaptation	Offline

Evaluation [MIT LL – Missing verdict expression to evaluate CP2 test results]

Figure 15. Overview of CP2 evaluation.

4.2.2 Virtual Machine Specification

The Virtual Machine (VM) Specification outlines the baseline software and hardware relevant environment as delivered by the platform team as well as that used during evaluation.

[AR – copied from MIT wiki]

IMMoRTALS will use one VM for the DAS, and nominally one VM for the server and each client process that is part of the platform. There are a variable number of client processes. The DAS and platform server VMs are assumed to be x86_64, and running modern linux. We are planning on using x86 Android emulators as well, because the ARM-based emulators running on x86 are so slow that they drastically affect the running applications. Our current Android client is targeting Android version 6. We will need two networks: a control network for deploying the applications and an experiment network for the platform applications to talk to each other. Initially for the 'offline' scenarios a single network will suffice.

TABLE 6

Virtual Machine Specification

	Delivered	Evaluated
Processor	single core x86_64	TBD cores x86_64
Memory	4 GiB	same
Disk	30 GiB	same
Operating System	Ubuntu 16.04 64-bit	same
Other Software Requirements	Android SDK and build environment; Java JDK 1.8	same

4.2.3 Test Harness API

This section describes the platform and challenge problem specific *Test Action Messages* as defined in the BRASS Phase 1 Communications API document [1]. Further communication channels supporting *Test Control Messages* for test orchistration, *Test Data File*, and *DAS Log File* for test configuration and logging are to be implemented exactly as described in the BRASS Phase 1 Communications API document and will not be detailed here.

[AR – copied from MIT wiki]

The mechanisms for perturbing the system for this problem consist of constructing a new set of mission requirements and execution environment descriptions. The number of clients, PLI report rate, and Image Report Rate are integer values that can be set to any positive value. Likewise the bandwidth availability will be given as an integer value (likely kbps). We will provide a set of image properties and their possible values. We will also provide a pre-populated repository of code fragments.

The evaluation system must be able to limit bandwidth for at least one link (typically will be the link to/from the server). Multiple android "clients" will need to be deployed according to the mission plan.

[AR – copied from Interface API section of MIT wiki post]

Our DAS implements a single REST call that takes a parameter in the form of a TTL (Terse RDF Triple Language) file. The TTL file describes the mission requirements and desired deployment options, including descriptions of the deployment hardware, software, and resource constraints, such as bandwidth or the availability of external services. The TTL file contains much more detail than is required for our Phase 1 Challenge Problems, so we intend to provide a script

that will allow easy tweaking of the parts we are addressing in Challenge Problems 1 and 2, while filling in the rest of the model from a template.

Between CP1 and CP2 we have a total of 9 possible dimensions of change. Some of these dimensions are a simple binary choice (such as whether the target Android device has bluetooth capability), and some are numerical ranges (e.g. positive integers for number of clients). The script and resulting TTL file will contain all the settings from both CP1 and CP2, so both challenge problems can be evaluated with the same infrastructure. If there is a desire to evaluate CP1 and CP2 independently, we will provide some nominal settings when evaluating CP1 to use for the CP2-specific dimensions and vice-versa (e.g. the dimensions that only apply to CP2 don't change when making perturbations for CP1).

Once a TTL file is obtained, whether using our script or another method, an HTTP POST call is made to our DAS with that file as a parameter. When that HTTP POST call returns, assuming it returns a SUCCESS code, new application binaries should be available for deployment and scenario execution.

[AR – the API definition is not yet populated.]

[Note – API to be updated.]

4.2.4 Parameter Specification

Listings 2 documents and describes/defines the parameters used in this challenge problem to perturb the ecosystem in which the DAS will be evaluated. Each parameter is defined in the format described in the MIT LL ParameterDSL document [2] and captures 1) parameter name 2) baseline value 3) constraints to be satisfied when choosing new values and 4) comments providing more information about each parameter.

The following definition provides guidance on the structured format for defining the parameters using the MIT LL ParamDSL.

```
// Comment
OptionalNamespace : {
    IdentifierName1 : baseline_value1 , constraint1;
    IdentifierName2 : baseline_value2 , constraint2;
    ...;
};
```

The baseline values defined for each parameter alway represent the conditions used when evaluating the system as delivered in its baseline state.

Listing 2: Challenge Problem 1 Paramters

```
// Shared network capacity
bandwidth : 1000, range (0, 10000000);

// Number of client devices
count: 2, range (2, 20);

// Time between SA reports (in milliseconds)
latestSABroadcastIntervalMS: 2000, range(1000, 60000);

// Time between image reports (in milliseconds)
imageBroadcastIntervalMS: 1000, range(1000, 60000);
```

4.2.5 Test Procedure

[Note – This section should define and describe the goals of each test case(s), trial, test suite, etc. necessary to evaluate the platform.]

4.2.6 Automata

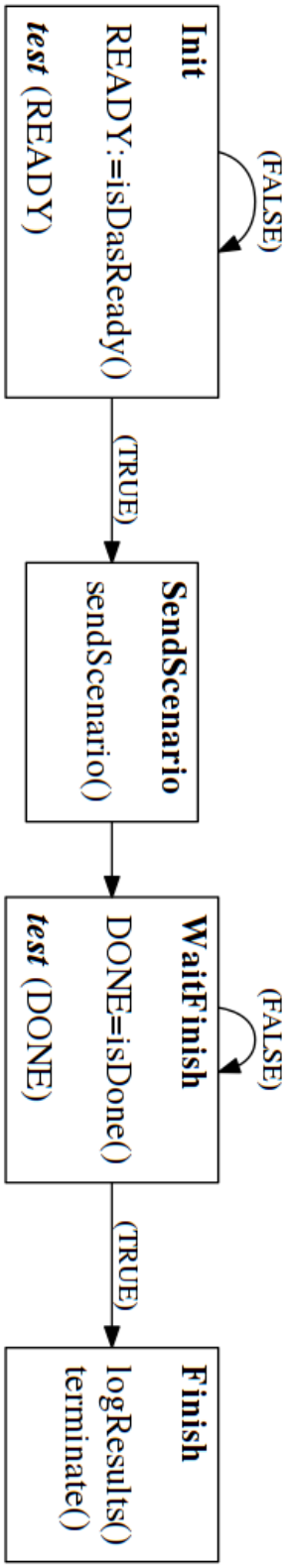


Figure 16. CP2 Automata.

TABLE 7

Functions used in the CP2 Automata

isDasReady() :=
    GET /DASSTATUS == Operational

startTest() :=
    POST /start_challenge_problem

4.2.7 Intent Specification and Evaluation Metrics

[MIT LL – Missing Intent element for CP2] **Element1**

TABLE 8

CP2 Intent Element 1: Element1

Intent: Element1	
Informal Description: Description.	
Formal Description: Description.	
Test Measurement: Test.	
Result Expression: (a, b)	
Verdict Expression:	
Mission Evaluation	
a ≤ b	Complete
a > b	Not Complete
???	Inconclusive
???	Invalid
???	Error
Quantitative Evaluation (Baseline vs. SUT)	
condition1	Enhanced Capability
condition2	Similar Capability
condition3	Degraded Capability
Mission Evaluation ≠ Complete	Inconclusive

4.3 EMPIRICAL EVALUATION

[Note – This section to be populated with key results by the Platform Workshop. More description and refinements to be made after the Platform Workshop.]

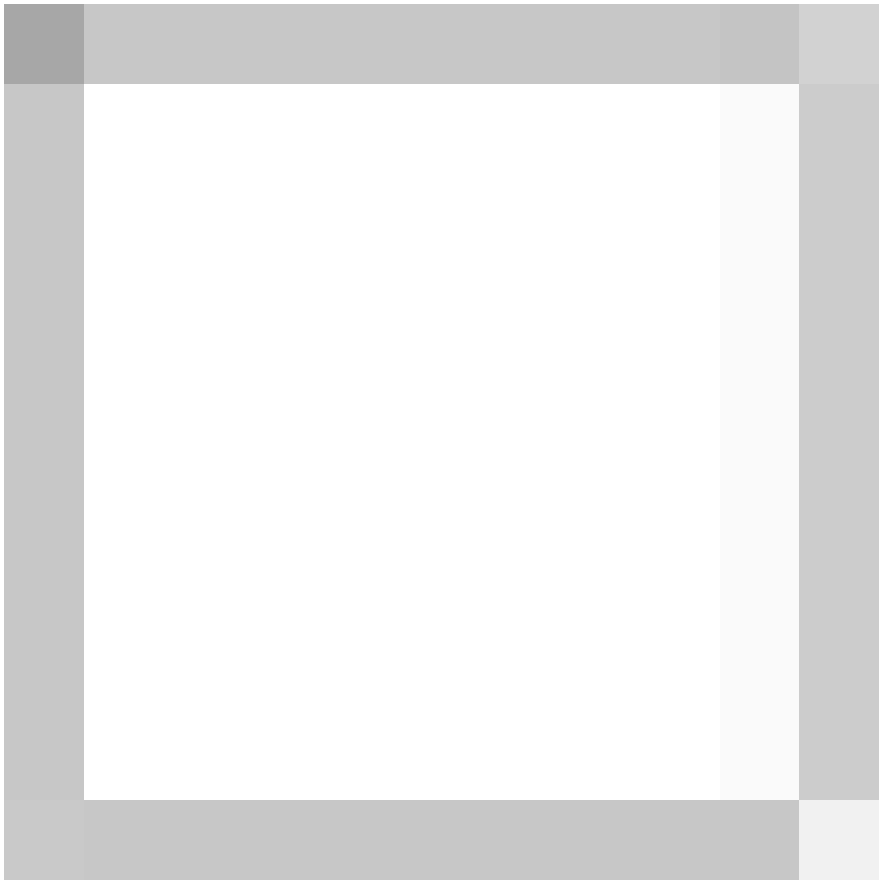


Figure 17. CP2 Summary Result

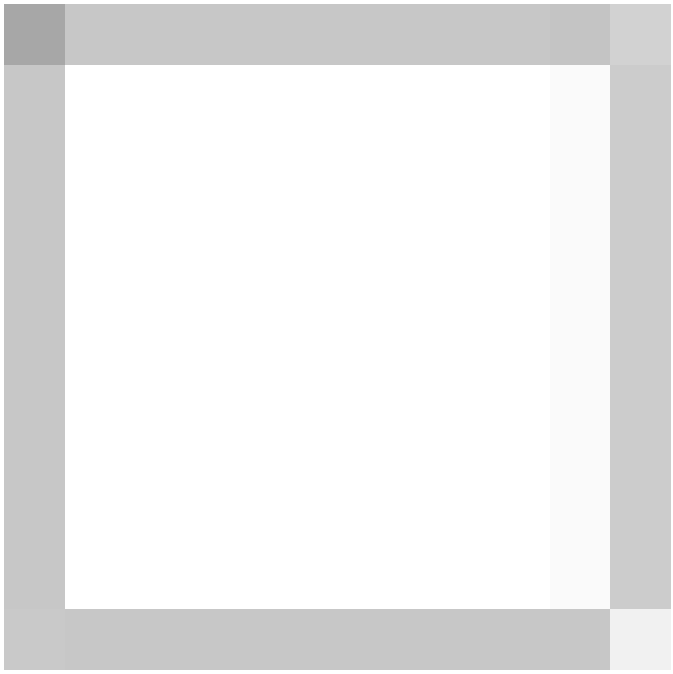


Figure 18. CP2 Key Result 1

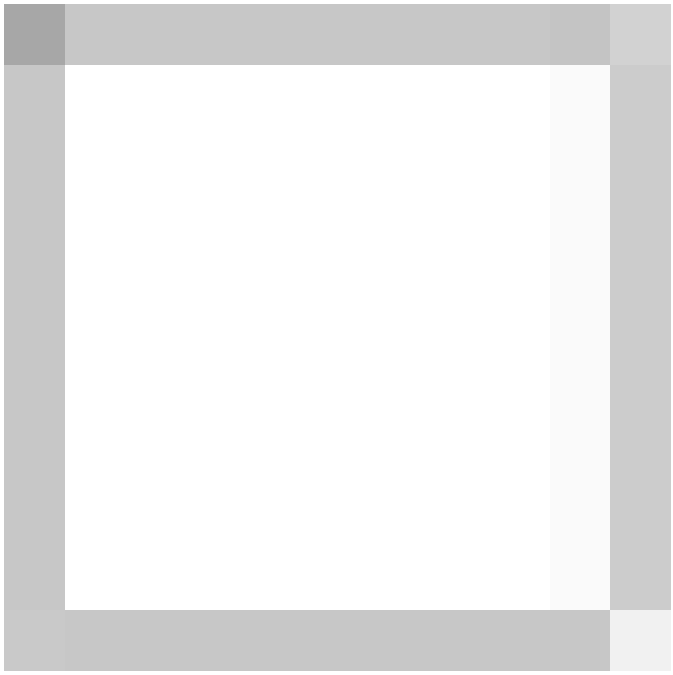


Figure 19. CP2 Key Result 2

4.4 DISCUSSION AND RECOMMENDATIONS

[Note – This section to be completed after the Platform Workshop.]

4.5 OVERALL ASSESSMENT

[Note – Discuss strengths, weaknesses and suggestions for improving this challenge problem]

4.5.1 Intent Discovery

Correctness

Completeness

Specificity

4.5.2 Analysis and Detection

Sensativity

Stability

Performance

4.5.3 Adaptation

Fidelity

Target System Performance

DAS Performance

4.5.4 OPTIONAL: Platform Specific Criteria

Expressivness of Specification Language

DAS Learning

Software Development and Maintenance

Impact Analysis

System Resilience

This page intentionally left blank.

5. OVERALL DISCUSSION AND RECOMMENDATIONS

[Note – This section to be completed after the Platform Workshop.]

5.1 OVERALL ASSESSMENT

5.1.1 Strengths

5.1.2 Weaknesses

5.1.3 Suggestions for Improvement

This page intentionally left blank.

REFERENCES

[1] MIT Lincoln Laboratory, "API Document," (Nov 15, 2016), URL https://wikis.mit.edu/confluence/display/BRASS/Documents, Ver. 0.7.

[2] MIT Lincoln Laboratory, "The Language ParamDSL," (Apr 19, 2016), URL https://wikis.mit.edu/confluence/display/BRASS/Documents.