AFRL-IF-RS-TR-2005-403
Final Technical Report
December 2005

HUMAN SYSTEMS MODELING AND SIMULATION

Synergia, LLC

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 074-0188*

1. AGENCY USE ONLY (Leave blank)	**2. REPORT DATE** DECEMBER 2005	**3. REPORT TYPE AND DATES COVERED** Final Jul 03 – Jul 05

4. TITLE AND SUBTITLE HUMAN SYSTEMS MODELING AND SIMULATION	**5. FUNDING NUMBERS** C - F30602-03-C-0099 PE - 62702F PR - 459S TA - MA WU - 01
6. AUTHOR(S) Gregg Courand and Michael Fehling	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Synergia LLC 2400 Broadway Suite 203 Redwood City California 94063	**8. PERFORMING ORGANIZATION REPORT NUMBER** N/A
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/IFSB 525 Brooks Road Rome New York 13441-4505	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER** AFRL-IF-RS-TR-2005-403

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Paul Yaworsky/IFSB/(315) 330-3690/ Paul.Yaworsky@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.	**12b. DISTRIBUTION CODE**

13. ABSTRACT *(Maximum 200 Words)*

New foundations for simulation studies of human systems have been created by: advancing the expressive adequacy of formal representations of human systems: treating physical properties and motion dynamics of human systems, along with the cogs five and social inter-dependencies that underwrite human behavior: designing, prototyping, testing and delivering extensions to Synergia's ACCORD technology for the computational specification of human systems (individuals and organizations): creating human-systems case studies to test this technology: developing a training program in Practice Mapping: assisting trainees with the Practice Mapping methodology and the creation of valid human-systems models for elements of a selected organization, so that the models support formal simulation studies.

14. SUBJECT TERMS Human Systems Modeling, Human Systems Simulation, Generative Practice Theory, Practice Mapping, ACCORD Tool Suite, Ambiguity Reduction Architecture, Actor Architecture, Organizational Modeling	**15. NUMBER OF PAGES** 186
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	**18. SECURITY CLASSIFICATION OF THIS PAGE** UNCLASSIFIED	**19. SECURITY CLASSIFICATION OF ABSTRACT** UNCLASSIFIED	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 **Standard Form 298 (Rev. 2-89)**
Prescribed by ANSI Std. Z39-18
298-102

Table of Contents

Executive Summary

We report our results on our research project, Human Systems Modeling and Simulation (HMS). This project has created new foundations for simulation studies of human systems. The primary objectives of the project have been as follows:

- Create a major advance in expressive adequacy of formal representations of human systems. Treat not only the physical properties and motion dynamics of human systems, but also and primarily the cognitive and social inter-dependencies that underwrite human behavior.
- Develop technology for the computational specification of human systems, based on the foregoing. Create human-systems scenarios to test this technology.
- Train AFRL-selected staff in Practice Mapping. Assist trainees in the mapping of an AFRL-selected organization, and the creation of valid human-systems models for elements of the selected partner organization. Support the trainee's development of simulation studies relevant to the mission or self-understanding of the organizational partner.

Our work is described in detail in the sequel. Here, we summarize how we have accomplished each objective.

Entering this project, we had previously specified practices, adaptive behavior generators, in terms of what we now call the Practice Proposition, "In Situation S, Do Activity A to Enforce Criteria C." We had proposed that a computational analogue, now called the Practice Schema, could be developed based on the idea of monitoring criteria-relevant conditions and directing computation accordingly.

We have now fully specified Practice Schemata. We describe them informally as: "Monitor for Conditions C in Situation S and Adjust the Execution of Activity A to Obviate C". In other words, criteria are served by monitoring for some conditions that reveal the criteria are not satisfied, then acting to eliminate or make irrelevant the criteria. When this occurs, the monitor is satisfied, and the practice is quiescent (suspends successfully, or fails).

We describe Practice Schemata formally in terms of a language specification we developed and report on here, called the "Quasi-Executable Practice Specification Language." This rigorous computational formalism[1] defines the syntax and semantics for specifying a practice schema's three components: an information environment (the situations created by and relevant to the practice), an activity generator (the *advisable* procedural content of the practice) , and a set of monitors (the tests that determine whether execution needs to be adjusted and if so, that suspend and restart the activity generator with new instructions. Triggering (i.e., initiation of a practice) is accomplished by monitors – behavior is in response to the failure of criteria.

We refer to the specification as a "quasi-executable" language because we aim to support simulation of human systems, within a variety of existing and to-be-developed simulation systems. It is beyond the scope of this project to develop a simulation environment or driver, or even, to forge the link between our formal specification and any particular environment or driver. The quasi-executable specification is complete with respect to the requirements of a practice, and a simulation driver could certainly be developed to manage simulation over a set of practices – i.e., practices defined in this language will execute if and once such a driver is built. However, the requirement for this project has been to sufficiently specify practices, in computational form, so that they *can be translated into a form compatible with* the native commitments of suitable existing simulation environments. We have achieved this.

In addition to the representational commitments of simulation environments, there are generally a host of other commitments having to do with expectations for the kinds of behavior to be simulated, and the kinds of studies/experiments to be conducted. For example, most simulation environments focus on the physical realization of actor state (i.e., location, resources, other properties). Actor behavior is often simplified to state transition (e.g., motion dynamics). Interaction (engagements) of actors in these simulations is not often modeled. Rather, a constructive model of the process is replaced with (perhaps complex) functions that determine the outcome of the interaction directly – e.g., as a function of the physical configuration of the participants (the opposition of their states). For example, such environments have inter-visibility

[1] Independently of this project, a fully algebraic formalization of Generative Practice Theory – and so for practices and actors – has been developed in the Pi Calculus. This work was a PhD dissertation (Stroh, 2004) at Stanford, supervised by Dr. Michael Fehling.

calculations that determine whether, when, and how actors may see one another's emanations. A simulation environment of this form might then favor experiments that focus on motion and compute results of engagements via attrition models.

Hence translation from quasi-executable practice schemata to the native form of a particular simulator entails not just reformatting (aligning representation) but also restriction to the range of behavior and experiment types expressible in the target simulation environment.

This means that there is a great deal about human system behavior that we can now express in our quasi-executable language, but that standard environments cannot fully exploit. Ideally, a simulation driver can be developed to run our models as fully specified. Until then, we find there is still great value in developing our models and converting them to more restrictive simulation environments. For example, we may model the risk attitude of an actor – a personality feature of that actor – and specify how the actor's current assessment of the proximity and severity of perceived threats affects the propensity to hide out versus move (depicted parametrically). In other words, a motion-dynamics-oriented simulation environment can still enjoy benefits of a validated model of an actor's personality, even if the simulation experiment cannot treat that personality directly.

In addition to this practice specification language, we developed and through several iterations refined an actor architecture, called the "Ambiguity Reduction Architecture." The Ambiguity Reduction Architecture (ARA) establishes the mechanism for practices to control one another. Whereas the practice provides for adaptation at the level of a given set of criteria, the ARA provides for adaptation across practices. In this way the *actor* (i.e., not just the practice) can intervene on its behavior as needed – as external conditions require, to resolve problems prior to continuing, to do something more important, etc.

The ARA is based on the recognition that there will often be conflicts among practices that need to be resolved – and therefore, ambiguity about 'what to do next.' These conflicts arise due to the inconsistency of practices (incompatible or context-specific learning trajectories lead to incompatibility), the lack of enabling resources for otherwise-mandated practices, or due to the finite cognitive capacity any actor. In each case, the ARA provides for suspending conflicting practices, or recognizing patterns of suspended practices that need to be resolved, and developing and legislating a resolution.

Ambiguity reduction is just another kind of practice – in situation S (conflict), do activity A (resolution procedure) to enforce criteria C (prevent/maintain/achieve what is most worthwhile in this situation of conflict). As such, the ARA provides that some practices will be supervisory with respect to others. This means that they will compute mostly over execution state of other practices, and their effect will be to resolve problems of execution state.

In the final version of the ARA, we distinguish four categories of practices: Reactive, Knowledge, Execution, and Deliberative. Reactive practices are sensor practices (i.e., 'drivers') and control practices that resolve ambiguity and otherwise manage these. Execution practices are effector-drivers and control practices that supervise them. Knowledge practices are those that couple the reactive and execution practices, along with their associated supervisory practices (i.e., those that resolve ambiguities by executing a pre-existing method, as a choice or policy). Finally, Deliberative practices are higher-level practices that form new ambiguity reduction methods and enter them as supervisorial with respect to other practices. Each category of practice is specifically identified with resources, e.g., sensors, tools, and cognitive capacity.

Practices may contain other practices as constituents. Therefore, the ARA may model (enables a modeler to specify) an individual human being, or any collective whose practices are inter-dependent. For example, if an individual (or a team) is gathering information, they may function as a Reactive practice within the ARA. Similarly, the actor (individual or social) that implements a COA is encapsulated as an Execution practice.

We prototyped technology to specify Practice Propositions, Practice Schemata, and Actors. We describe these in the sections to follow. We integrated this fully within our ACCORD environment. Hence the new tools inter-operate with the extensive technology we have developed to gather and formalize data on human behavior. In addition, we have developed (on other work) a Windows-based and a Java-based API to enable other technologies to inter-operate with ACCORD. Therefore, we have developed the necessary technology to deliver quasi-executable human-systems models.

We are delivering the technology prototyped on this contract within the complete ACCORD environment. Hence we are delivering the tools that support specification of computational models of human systems, along with the most recent versions of all other tools, including those developed on other projects[2].

We conducted a substantial amount of scenario development work, to test the quasi-executable specification and our practice and actor specification technologies. One scenario is a moderately detailed mapping study in which a Key Figure is seeking to escape to safe haven. We do not see this individual, but we see the effects of his behavior on others. This scenario emphasizes the capture and formalization of data (100 days of message traffic on our pursuit of him), and then the methods that lead to discovery and articulation of practices of that actor. In this scenario, our emphasis has been to illustrate the overall disciplines of mapping (gathering data, creating a suitable ontology, formalizing in event-behavior graphs) modeling (creating practice and actor models), and analysis (introducing forecasts into the evolving event-behavior graph).

We also invented a multi-actor case study, this time to support a comprehensive simulation study. We developed all of the practices of all of the actors sufficiently to provide for actor sensing (self and others), deliberation, behavior, and joint (interactive) behavior. This scenario is also based on a manhunt – a narcotics trafficker seeks to escape to safe haven.

We specified the latter case study, the actors and their practices, using our quasi-executable language specification, and embedded in the tools for the ambiguity reduction architecture. We are delivering this case study as a project within ACCORD. We also specify the independent variables and so the range of potential simulation studies/experiments that might be run.

[2] AFRL funded an allied project, now completed, to create elements of simulation support needed to manage the actor models and simulation studies that this project provides for. We report those results separately. In summary, we:

- developed a formal foundation to characterize the uncertainty of a model of an actor (as a function of uncertainty in practice models, and thus on the evidence and assumptions that warrant the models)
- developed technology to manage 'releases' of practice and actor models (users publish a model that is fully linked to its evidential and assumptive foundations; models are maintained in a history and browseable)
- developed the necessary indexing scheme to manage a collection of related simulation studies (and so specify independent variable bindings, actor model version, and dependent measures).

We conducted a variety of training initiatives on this contract. Early in the project, we produced a book-length monograph on Practice Mapping methodology, and delivered it to AFRL.

We invented and prototyped a new tool, called the Journal tool, to guide trainees in Practice Mapping to become more careful about their observations, and to begin to make important cognitive and methodological distinctions. The trainee made use of this method, and his results supported extensive methodological training based on the human-system activity he observed.

We developed a comprehensive curriculum for trainees. This curriculum includes background studies in Decision Theory, extensive exercises to learn Generative Practice Theory by working concrete examples (using Synergia-supplied data), and guidance for mapping the practices and organizational structure of a partner organization (e.g., an Air Force planning or mission management component).

We spent two days with the trainee, presenting Generative Practice Theory (GPT) and the Practice Mapping (PM) methodology. We also worked with the trainee on a collection of exercises we designed and he worked on, prior to joining us for GPT and PM training.

After this, we conducted ongoing training, following our usual methods. We reviewed trainee products on exercises. We provided extensive instruction in GPT and PM, in the form of critiques, extensions, and instruction on the trainee's submissions. The curriculum, and a sample exercise and associated Synergia instruction appear in this report.

We delivered a copy of ACCORD to the trainee, to practice with, and in preparation for that trainee to map an organization of interest.

AFRL leadership worked repeatedly over the life of this project to identify a suitable organization to map (e.g., a part of a JFACC), and this formed a topic of discussion at each meeting. Synergia supported this activity by gathering material and learning what we could about each candidate identified by AFRL, and evaluating mapping requirements should that candidate agree to participate. Synergia also identified, contacted, and attempted to engage suitable Air Force partners (e.g., Checkmate, Air Force Studies & Analysis Agency). For these Synergia-developed candidates, AFRL visited senior staff in the Pentagon on our behalf.

However, for a variety of reasons, including complexity of candidate organizations, ongoing reorganization initiatives, strong conflicting demands (due to the ongoing missions in Afghanistan and Iraq), and the lack of any mandate requiring those organizations to participate, an organization willing to serve as a partner on this work was never identified. Hence the trainee did not map an existing organization.

Given the lack of a partner organization on which to base mapping and then simulation studies, AFRL ultimately decided to have the trainee focus on learning the JSAF simulation environment. Once we learned of this decision, we reorganized our effort to produce the most extensive simulation study we could accomplish on our own. We concentrated on the development of the latter simulation case, and ensured that it fully exercised the actor architecture – in particular, requiring close coupling among sensor, knowledge, and execution practices, within and across actors. Given the selection of JSAF, we chose to produce a simulation exercise with significant motion-based content. We fully specified the actor models in quasi-executable form, and are delivering them in this report and as an ACCORD library. We are confident that this modeling exercise underwrites useful and interesting investigations of quasi-executable human-system simulation, and we look forward to future work with AFRL, in which we would advise on and pursue such investigations.

The sections that follow present detailed results on the quasi-executable practice specification language, actor architecture, and training. For a review of the theoretical foundation provided by Generative Practice Theory, please see Appendix A. Appendix B contains a sample practice from the first scenario, and Appendix C the simulation specification for the second case study we invented (also delivered within ACCORD).

1. Quasi-Executable Specification Of Human Systems as Systems of Practices

We present an overview of practice schemata and their intended form of execution. Next we describe the formal specification of the computational form of practices, namely, "practice schemata". Then we describe the actor architecture we designed, so that individual and social actors can be implemented as collections of interdependent practices. We implemented an interface to support the definition of practices, and the basic actor architecture on this contract – the ability to assemble individual actors as systems of practices, assigning the practices as one of: sensory, base, supervisory, and effector.[3]

We now turn to the practice construct. Recall than an actor is a system of practices, which is a practice. Hence this construct can be read as the generative capacity for skills, individuals, and social formations.

Practice Schema

A practice schema is a computational model of a practice. Recall that a practice is defined by the proposition: In Situation S, Do Activity A to Enforce Criteria C.

A practice schema is composed of an information environment, a set of activity generators, and a set of monitors.

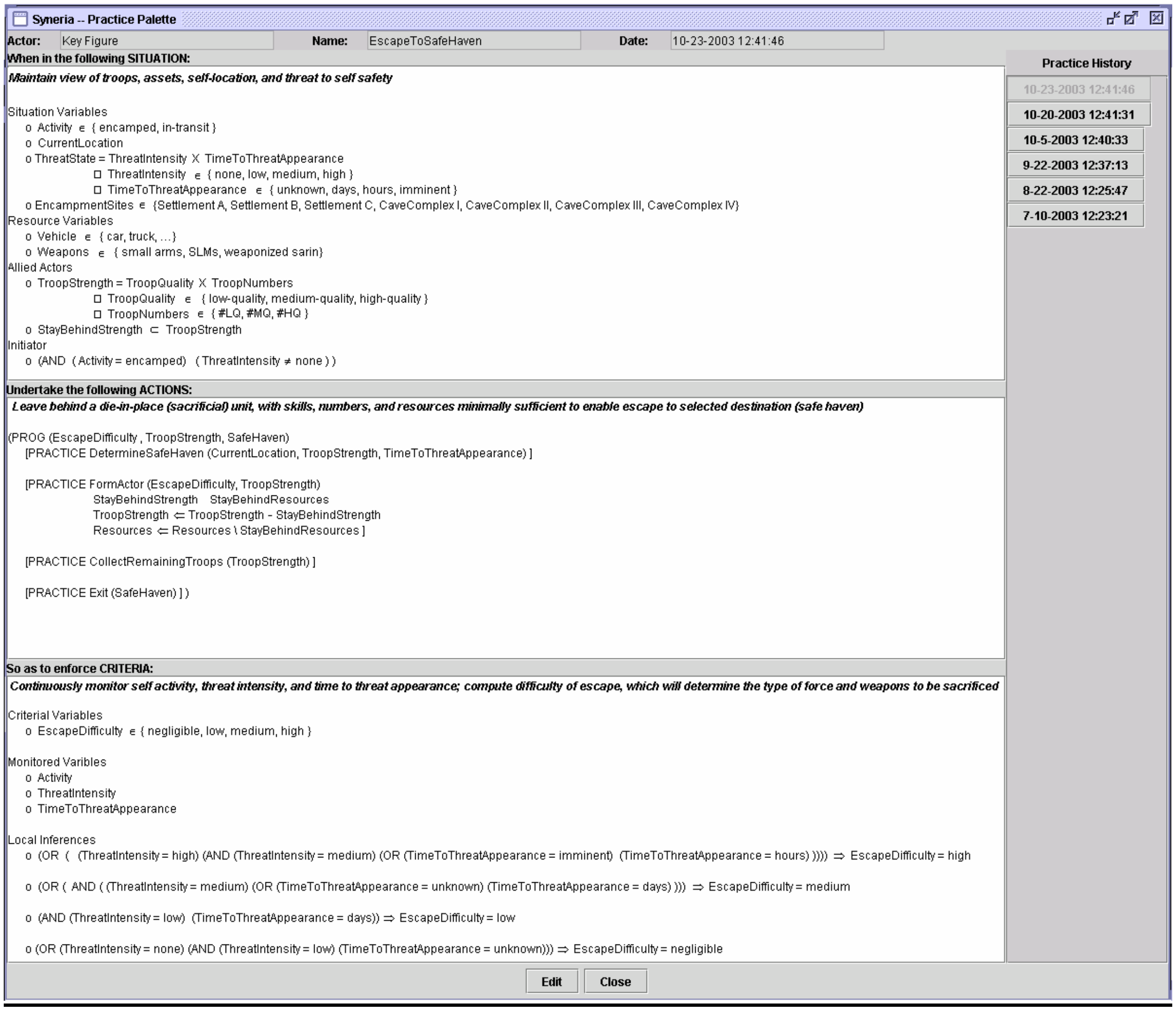
The function of a practice schema is to generate a computational image of a behavior – a computational trace that models human behavior as responding to initiating and in-process events, and strives to enforce criteria associated with that practice.

Criteria are not explicitly reified in practice schemata – there is not a component that contains the criterion. Instead behavior is shaped by the information, activity generator, and monitors *so that the criteria tend to be enforced.* (Criteria are enforced if conditions permit, and so long as more important criteria do not intervene). Typically, a practice has some reference for what counts as good or better, and behavior is managed by monitoring for conditions that indicate the

[3] We are developing the structures that enable formalization of actor interdependencies on another contract; a first version is in place.

reference is unmet and then signaling the activity generator to compensate – with the signal reflecting the type and scale of the error (departure from the reference). We often call the reference a “governing variable.”

We show a sample practice schema here, within ACCORD. Note the separate definition of situation, activity, and criteria (monitors), the maintenance of ancestor, descendant, and variant practices, and the evidence and rationale that warrant this new model of the practice.



Actor:	Key Figure	Name:	EscapeToSafeHaven	Date:	10-23-2003 12:41:46

When in the following SITUATION:

```
Maintain view of troops, assets, self-location, and threat to self safety

Situation Variables
    o Activity ∊ { encamped, in-transit }
    o CurrentLocation
    o ThreatState = ThreatIntensity X TimeToThreatAppearance
                    □ ThreatIntensity ∊ { none, low, medium, high }
                    □ TimeToThreatAppearance ∊ { unknown, days, hours, imminent }
    o EncampmentSites ∊ {Settlement A, Settlement B, Settlement C, CaveComplex I, CaveComplex II, CaveComplex III, CaveComplex IV}
Resource Variables
    o Vehicle ∊ { car, truck, ...}
    o Weapons ∊ { small arms, SLMs, weaponized sarin}
Allied Actors
    o TroopStrength = TroopQuality X TroopNumbers
                    □ TroopQuality ∊ { low-quality, medium-quality, high-quality }
                    □ TroopNumbers ∊ { #LQ, #MQ, #HQ }
    o StayBehindStrength ⊂ TroopStrength
Initiator
    o (AND ( Activity = encamped) ( ThreatIntensity ≠ none ) )
```

Undertake the following ACTIONS:

```
Leave behind a die-in-place (sacrificial) unit, with skills, numbers, and resources minimally sufficient to enable escape to selected destination (safe haven)

(PROG (EscapeDifficulty , TroopStrength, SafeHaven)
    [PRACTICE DetermineSafeHaven (CurrentLocation, TroopStrength, TimeToThreatAppearance) ]

    [PRACTICE FormActor (EscapeDifficulty, TroopStrength)
                    StayBehindStrength   StayBehindResources
                    TroopStrength ⇐ TroopStrength - StayBehindStrength
                    Resources ⇐ Resources \ StayBehindResources ]

    [PRACTICE CollectRemainingTroops (TroopStrength) ]

    [PRACTICE Exit (SafeHaven) ] )
```

So as to enforce CRITERIA:

```
Continuously monitor self activity, threat intensity, and time to threat appearance; compute difficulty of escape, which will determine the type of force and weapons to be sacrificed

Criterial Variables
    o EscapeDifficulty ∊ { negligible, low, medium, high }

Monitored Varibles
    o Activity
    o ThreatIntensity
    o TimeToThreatAppearance

Local Inferences
    o (OR ( (ThreatIntensity = high) (AND (ThreatIntensity = medium) (OR (TimeToThreatAppearance = imminent) (TimeToThreatAppearance = hours) )))) ⇒ EscapeDifficulty = high

    o (OR ( AND ( (ThreatIntensity = medium) (OR (TimeToThreatAppearance = unknown) (TimeToThreatAppearance = days) ))) ⇒ EscapeDifficulty = medium

    o (AND (ThreatIntensity = low) (TimeToThreatAppearance = days)) ⇒ EscapeDifficulty = low

    o (OR (ThreatIntensity = none) (AND (ThreatIntensity = low) (TimeToThreatAppearance = unknown))) ⇒ EscapeDifficulty = negligible
```

Practice History

- 10-23-2003 12:41:46
- 10-20-2003 12:41:31
- 10-5-2003 12:40:33
- 9-22-2003 12:37:13
- 8-22-2003 12:25:47
- 7-10-2003 12:23:21

Information Environment

The Information Environment is defined by a list of variables, and their domains, that influence the practice. Some variables are internally generated; others are communicated from other (external) practices.

The information environment may save state as a result of execution, accessible under future execution – as when a behavior recurs or is suspended and restarted, and the later execution depends on elements of the prior one.

Monitors compute over the contents of the information environment.

Activity Generator

The Activity Generator is a set of five functions, *initialize*, *main*, *pause*, *resume*, and *kill*, used to describe the actions taken by an actor to enforce certain criteria. The bodies of each of these functions are defined using the language constructs and basic data types described in our language definition.

The *initialize* function is used to set the internal variables of the practice, which are used by the Activity Generator's *main* function.

The *main* function is the definition of the action steps executed by the actor. This function is not only comprised of our language constructs and basic data types as mentioned before, it can also contain references to other practices. This function is invoked by monitors when they establish that certain criteria are not being met.

Note that the *main* function may embody parallel generators, and also may embed a set of practices. It is not limited to being a single function.

The *pause* function is used to interrupt the Activity Generator's *main* function. This function stores its variables' states, as well as the point in the function at which it was paused.

The *resume* function is used to restart the Activity Generator's *main* function at the point it was paused. It uses the information that was stored by the function *pause* to do so.

The *kill* function is used to terminate the Activity Generator's *main* function before its natural completion. It allows the user to define the proper terminating procedure steps to take.

Monitors

Monitors consist of:

- a list of variables (the "event list") that are monitored for changes,
- a specified condition on those variables (the "decision schema"), and
- a procedure body that accomplishes the control action that advises or redirects activity generators.

Monitors may have the form of busy-wait tests, those that are active while monitoring, and may also have the form of non-busy-wait tests, those that are not active while monitoring for the lack of satisfaction of criteria.

The event list may consist of variables communicated from outside of the practice, internally-generated variables, or both. However, regardless of the source, the information is within the information environment of this practice.

The decision schema generally tests "while" and "when" conditions on instances of the event list (subsets of the domains defined for those variables, i.e., specific bindings). The decision schema may compose tests according to logical operators.

The procedure body is comprised of our language constructs, basic data types, and references to practice Activity Generators. It may have the form of a case statement, selecting among or otherwise organizing the ensuing activity generators.

As the monitored variables change, they are compared to the decision schema. Positive matches launch the procedure body of the monitor, which interrupts execution of an activity generator and may then launch other activity generators, or relaunch the first one, under specified conditions. These 'return conditions' are determined as a function of the evaluated decision schema.

Summary of Practice Execution:

Practices are generators that exist to satisfy actor criteria.

Practices are controlled by monitors. Monitors test for evidence of failure to satisfy criteria. If so, they launch, suspend, or redirect computation to reify an actor's method of managing (adapting) behavior. Note that initiation of the practice is just another instance of monitoring.

Monitors may test directly for criterial satisfaction, or they may test conditions that imply something about criterial satisfaction. It is the responsibility of the activity generator to (strive to) repair the deviation from desired criteria.

When monitors detect changes in the states of variables they are monitoring, they evaluate the decision schema – to determine whether a change in the activation or execution status of activity generators is warranted. If so, they modify the appropriate execution status of the relevant activity generator(s). Note they can only operate on the execution trace 'so far'; monitors cannot redirect execution to registers corresponding to future (not yet effected) computation.

Concurrent with the process of procedure execution within activity generators, monitored variables may change. This may occur as a consequence of procedure execution, or due to external influence. These changes may create the need to interrupt and redirect the activity generator.

This interrupt and redirect protocol is accomplished by a form of continuation. It uses the *pause* and *resume* functions: the main procedure is temporarily paused to allow the monitor to determine (via its procedure body) how to redirect computation. This may await changes in conditions created externally or via other practices. It may resume from the point at which it was paused, or from an earlier point. It may use any information in the information environment that has occurred prior to and during the pause, as well as information created via execution thusfar. Pause can also lead to termination: the main procedure is aborted. This is done with a call to the *kill* function. Kill stops the main procedure, does some final steps, and then exits.

Program debuggers offer a useful analogy to practice execution. Once the debugger is launched it lives in the background monitoring for problems in the execution of a program. If a problem is detected it interrupts the program, does the work it needs to do to display errors and allow the user to make changes to the program, and then lets the program resume were it was interrupted. This is analogous to concurrent execution of monitors that may pause ongoing execution of an activity generator and then terminate it, or resume it at an earlier point and with revised state information.

1.1 Computational Specification of Practice Schemata

We present the formal specification language for quasi-executable practice schemata.

1. <u>BASIC DATA TYPES:</u>

TYPE = { Integer, Real, Boolean, String } ∪ SET_NAMES

2. <u>KEYWORDS:</u>

if, else, par, [when], while, break, continue, {, }, return, Integer, Real, Boolean, String, set, &&, ||, setdiff, subset, void, ";", activitygen, monitor, practice, typegen

3. <u>LANGUAGE CONSTRUCTS</u>

Triggering_Event_List := (ε | TYPE var_name [, TYPE var_name]*)

IF := **if** *CONDITION* { ([*ACTION*]+) }

IF-ELSE := *IF* **else** ([*IF-ELSE*] | ε) *ACTION*

WHEN := **when** *CONDITION* { ([*ACTION*]+) }

PAR := (**if** *CONDITION* | **when** *CONDITION*) **par** { Action-1 … Action-n }

ACTION := (Block_Of_Code **|** activityGen_name)*

AND := *CONDITION* **&&** *CONDITION*

OR := *CONDITION* **||** *CONDITION*

COMPARISON_OP := <, <=, >, >=, =, !=

COMPARISON := (var_name **|** *VALUE*) *COMPARISON_OP* (var_name **|** *VALUE*)

ARITHMETIC_OP := + **|** - **|** * **|** /

ARITHMETIC := (var_name **|** *VALUE*) *ARITHMETIC_OP* (var_name **|** *VALUE*)

WHILE := **while** *CONDITION* { ([*ACTION*]+) }

SETDIFF := dest_set := first_set **setdiff** second_set

/* text */ traditional comment

// text end-of-line comment

4. INFORMATION ENVIRONMENT

Set Declaration:

set set_name = { elem-1, elem-2, …, elem-n };

Variable Declaration:

TYPE var_name;

Variable Assignment:

var_name := <*VALUE*>;

SubSet Declaration:

subset set_name : another_set_name;

5. <u>ACTIVITY GENERATOR</u>

activitygen activityGen_name(*Triggering_Event_List*)

```
{
        PARAMETER_LIST_REFERENCES;
        PARAMETER_LIST_COPY;

        void Initialize(PARAMETER_LIST)
        {
                <initialize PARAMETER_LIST_REFERENCES>
                <initialize PARAMETER_LIST_COPY>
        }
        void Main()
        {
                <Procedure_Body>
        }
        void Pause()
        {
                <Procedure_Body>
        }
        void Resume()
        {
                <Procedure_Body>
         }
        void Kill()
        {
                <Procedure_Body>
        }
}
```

6. PROCEDURE

```
<return_type> proc_name([PARAMETER_LIST])
{
        [Built-up from language constructs]
        [LOCAL_VARS]
        [PROCEDURE_CALLS]
        [PRACTICE_REFERENCES]
}
```

7. MONITOR

```
monitor monitor_name(Trigger_list)
{
        while (true)
        {
                if (CONDITION)
                        activityGen_name.<method_name>
                else if (CONDITION)
                        activityGen _name.<method_name>
```

```
          else
                activityGen _name.<method_name>
    }
}
```

8. PRACTICE SCHEMA

```
practice practice_name
{
    [INFO_ENV]
    ([MONITORS]*)
    ([ACTIVITY_GENS]*)
}
```

9. TYPE GENERATOR

```
typegen type_name = { first_element, generator_function }
```

10. GENERATOR FUNCTION

```
type_name generator()
{
    return <expression>
}
```

1.2 Actor Architecture

Synergia invented our actor architecture, called the "Ambiguity Reduction Architecture," prior to this contract. For this project, we critically reviewed its contents, made a collection of improvements, and implemented a first version within our ACCORD technology.

The Ambiguity Reduction Architecture (ARA) establishes the mechanism for practices to control one another. Whereas the practice provides for adaptation at the level of a given set of criteria, the ARA provides for adaptation across criteria. In this way the actor can intervene on its behavior as needed – as external conditions require too resolve problems prior to continuing, to do something more important, etc.

The ARA is based on the recognition that there will often be conflicts among practices that need to be resolved – and therefore, ambiguity about 'what to do next.' These conflicts arise due to the inconsistency of practices (incompatible or context-specific learning trajectories lead to incompatibility), the lack of enabling resources for otherwise-mandated practices, or due to the finite cognitive capacity any actor. In each case, the ARA provides for suspending conflicting practices, or recognizing patterns of suspended practices that need to be resolved, and developing and legislating a resolution.

Ambiguity reduction is just another kind of practice – in situation S (conflict), do activity A (resolution procedure) to enforce criteria C (whatever is most worth preventing/maintaining/achieving in this conflict). As such, the ARA simply provides that some practices will be supervisory with respect to others. This means that they will compute mostly over execution state of other practices, and their effect will be to resolve problems of execution state.

We critically reviewed the Ambiguity Reduction Architecture over the course of this project, and in particular, our preliminary specification for its design. Our primary conclusions are as follows:

- In our initial design, we distinguished between supervisory practices and base-level practices. However, we did not provide a simple discipline for writing out information to which the supervisory practices may respond. We also do not have a general mechanism for practices to influence the execution of other practices, save by direct communication. These two factors were addressed by creating a general memory store that enables practices to write out information, such that others may detect relevant patterns and initiate or advise execution. This general memory store functions as a shared resource. This is why we earlier referred to "information within the scope of the information environment" rather than "information within the information environment." Monitors can monitor local information, as well as space within the general memory store to which they have been granted access/visibility.

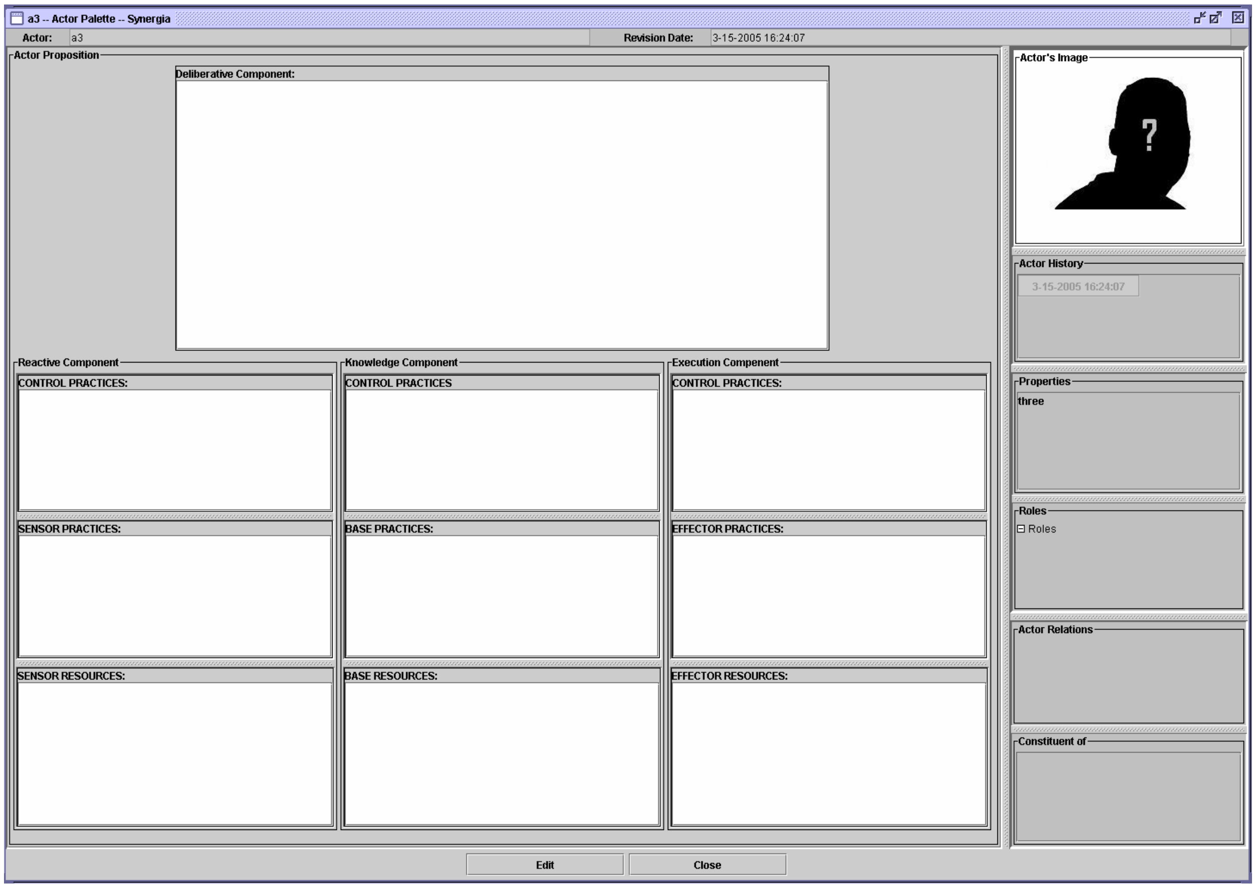
- Actors need an associated state, which can be modified under the execution of actor practices (self, others) and physical processes. In other words, actors have a history and a physical state that may be available to themselves and others. It is feasible but very inconvenient to attach this state to individual practices, rather than to the actor as a whole.

- Our original architecture focused strictly on practice encapsulation, and so provided for sensors, deliberative practices, supervisory practices, and effectors. In our studies using the architecture, we discovered that we have inadequate provision for modeling sensors versus sensory practices, and effectors, versus effecting practices. It is also useful to formalize the cognitive resources available to the actor – e.g., a model of the number of 'chunks' an individual can manipulate concurrently. Therefore, we decided to segregate practices as one of {sensor-driver, effector-driver, deliberative, supervisory} and make the resources on which each depends explicit.

- Resources are one of {physical artifacts, information objects}. Resources are implemented as a type of actor, embodying less or more intentionality – less for chairs and Word documents and copy machines, and more for actors functioning in the role of resources. The chief difference between a purely physical object and a human, using this approach, is that the human actor has the possibility, through reflection and learning, of changing criteria. For non-intentional actors, criteria are fixed unless and until a human changes them. At the extreme (chairs, rocks) objects are actors/practices in the degenerate sense that they have state, may be acted upon, and operate according to physical laws – but beyond this, do not generate behavior ‘on their own.’

We remark that the segregation of practices from the resources they depend on clarifies the use of the architecture to specify social actors. For example, it was clumsy before to refer to an individual as a sensor for an organization (e.g., a reconnaissance officer for a battalion, or a sales division of a firm). Now, we: create the organizational actor, install the relevant practices of the individual in the sensor-driver class of practices for the organizational actor, create the inter-dependencies between that individual and other individuals in the organization by modeling the other practices and their inter-dependence, and, install any actual physical sensor artifacts used by the individual in the associated sensor resource category. In this way the dependencies among actors are captured ‘on the effector side’ of the practices of the individual acting as sensor, and the management of resources are captured ‘on the sensor side’ of the practices of that individual. Of course, this all depends ultimately on the fact that practices or systems of practices may be constituents of other practices.

In our latest formulation, we segregated control (supervisory, ambiguity reduction) practices according to the primary categories (sensor, base, effector), and provided for a highest-level control regime. Thus we have the Reactive Component, the Knowledge Component, the Execution Component, and the Deliberative Component. The Reactive Component contains the Sensor Practices of an actor, the resources need by those practices and the Control Practices that resolve ambiguities amongst the Sensor Practices. The Knowledge Component contains the Base Practices of an actor, the resources need by those practices and the Control Practices that resolve ambiguities amongst the Base Practices. The Execution Component contains the Effector Practices of an actor, the resources need by those practices and the Control Practices that resolve ambiguities amongst the Effector Practices. The Deliberative Component describes the processes in which the actor partakes to develop Control Practices for resolving ambiguities. The image below shows the interface for this new Actor Architecture.


a3 -- Actor Palette -- Synergia

Actor: a3

Revision Date: 3-15-2005 16:24:07

Actor Proposition

Deliberative Component:

Reactive Component

CONTROL PRACTICES:

SENSOR PRACTICES:

SENSOR RESOURCES:

Knowledge Component

CONTROL PRACTICES

BASE PRACTICES:

BASE RESOURCES:

Execution Compenent

CONTROL PRACTICES:

EFFECTOR PRACTICES:

EFFECTOR RESOURCES:

Actor's Image

Actor History

3-15-2005 16:24:07

Properties

three

Roles

Roles

Actor Relations

Constituent of

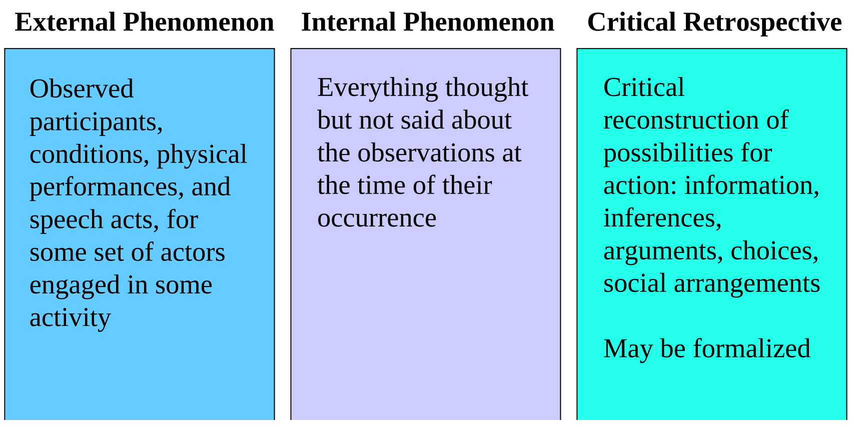
2. Practice Mapping Training Program

Human-systems modeling and simulation depends on the ability of a collection of analysts to gather and formalize data on human-systems behavior, based on Generative Practice Theory. Even analysts that wish to simulate human systems using practice schemata need to understand the theory and methods, though not as deeply as those who will gather behavior data and formalize it as actors (practice schemata within the ARA). We describe our activities here.

We developed an initial but detailed training program for prospective mappers. This program included lectures, as well as a carefully designed set of exercises. We begin with a method we developed to help trainees better characterize their observations of human activity. Then we present the curriculum for the exercises, followed by an example training interaction.

2.1 Developing Effective Observation Skills

We developed a new technique to support initial training in behavior observation and critical assessment of the observations. The goals are to help mapper trainees begin to observe behavior with as little bias as possible, to give them experience distinguishing between what is inter-subjectively visible and what is thought, and to lead them into the development of a productively critical stance with respect to the witnessed behavior.


External Phenomenon
Internal Phenomenon
Critical Retrospective
Observed participants, conditions, physical performances, and speech acts, for some set of actors engaged in some activity
Everything thought but not said about the observations at the time of their occurrence
Critical reconstruction of possibilities for action: information, inferences, arguments, choices, social arrangements
May be formalized

We illustrate its general form here. In essence, users have three panels for entering material on behavior observations: a base narrative to the left contains all externally-visible observations of the situations, actions, and communication of some human system engaged in some activity; a central panel in which the user describes everything that she/he thought as the observed activity transpired; and then to the right a critical reconstruction of the premises and larger potential implied by the observed activity (often formalized as a choice that encompasses what actually transpired), as well as any views on how observation or thoughts about the activity might have influenced the activity.

We developed a tool within ACCORD that supports these actions, called the "Journal Tool." (We refer to this tool as the "Diary Tool" – its original name – in the training course curriculum that appears below.)

2.2 Practice Mapping Training – Course of Study

We developed a delivered a training course on Generative Practice Theory, Practice Mapping, and ACCORD, as part of this project. We describe this work below. First, we offer a few general remarks on what we are learning about training.

Synergia is receiving increasing numbers of requests for training in our theory, methods, and technology, and has now trained a substantial number of candidates, with widely varying backgrounds and application interests. A number of candidate trainees have been presented, and we are evaluating our experience, as a precondition to improving training.

We require computer science background for any trainee that will be developing simulation models. Beyond this, we have not issued specific requirements; training is meant to be comprehensive. In many but not all cases, we have had the opportunity to participate in the selection of the candidate for training.

We find that trainees vary in terms of native aptitude – by which we mean their sensitivity to cognitive and social phenomena, and in their ability to abstract and formalize observations of such phenomena. However, aptitude has not proven to be a barrier to success. For example, event-behavior graphs appear to be readily understood and with training and a little practice easily formed, regardless of aptitude.

We also find that trainee candidates vary in their desire to learn our theory, methods, and tools, with very distinct impact on the effectiveness of training. The first type of trainee expresses disinterest in undergoing training, prima facie disbelief on new approaches like ours, or even distaste for the effort required. The second type of candidate expresses optimism or, at least, curiosity regarding our methods and their usefulness. The first type of candidate[4] struggles and requires far more time and effort from the instructor than does the latter. The latter type of trainees has consistently produced useful models and analyses right away, and, on their own initiative, many of these have taken steps to promote the approach within and even outside of their organization.

After having trained a significant number of candidates – many more than on this contract – we find that well over two-thirds are in the latter/productive category. We find we only experience the first/difficult trainee when we do not get to participate in the selection of the candidate. Therefore, we conclude that a careful screening process is essential for training candidates to ensure that they are properly motivated and provided sufficient incentives to ensure an appropriate level and quality of effort.

We continue now with specimens of training on this contract.

2.2.1 Prefatory Remarks

There are three components of Practice Mapping training. You will be working on them in parallel. In order of increasing importance, and with my suggestions for the relative amount of time and effort you devote to each:

- 20% background studies on decision analysis,
- 30% learning the ACCORD tools, and
- 50% beginning to map a JFACC.

In the case of all of this work, call or send me questions at any time. If I'm busy at the moment of a call, we'll arrange a time to discuss or decide what to send me.

Of course, send or call with questions on this program.

Never set something aside because it doesn't make sense or because, in trying it, something didn't work out. Get my help. This is the only way I can learn what you need!!! More

[4] We have had candidates that opted out of training. Our remarks here apply to those who have taken the training.

importantly, you should know that a significant aspect of learning our work is learning to function as an apprentice. We train the most critical things by starting with whatever poses a problem.

Send me specimen products at the end of each week or two (don’t go longer than 2 weeks!!), so that I can see how it’s going. Send me things right away when you start a new activity – e.g., the first time you analyze a newswire story, send me your result so I can check in and help if needed.

2.2.2 Background Decision Analysis Studies

1. Begin learning decision analysis.
    a. Study chapters 1-4 Making Hard Decisions, (Clemen) 2nd edition.
    b. For chapters 1-4 you should read every problem and assure yourself you have a good feeling for what it is asking you to do. If you wish, work some portion of the problems.
    c. Don't worry about using the DecisionTools software; if you find yourself wanting to use tools, use ours instead. This will help us think about what we need to add or revise.
2. Optional: if you want to learn more about value, and value modeling, read chapters 1-5 of Value Focused Thinking (Keeney)

2.2.3 ACCORD Explorations

1. Use the diary tool once/day or at least a few times per week.
    a. Fill in the narrative and private-thoughts sections. We'll get to the third column (critical reflections on foregoing) in a while.
    b. Pick activities -- preferably multi-person ones -- in which participants were
        i. analyzing or figuring something out,
        ii. deciding something,
        iii. a conclusion was shared that is pertinent to some choice of one of the participants, or

    iv. a choice of one participant was transmitted to another (e.g., boss gives direction on what to do).

2. Analyze each of the newswire files you will find in appendix that follows. Begin by doing a "GPT markup" of each separate article.

    a. You can do this by creating a table for each article, if you wish – elements of the GPT vocabulary, and instances in the article. Or, you might put the article in PowerPoint and then color patches of text with a distinctive color for each kind of item (e.g., actors are orange, roles are blue, objects are green, etc.)

    b. Specifically, identify:

        i. **actors** (individuals, groups)

        ii. **behavior** and **behavior trajectories**: this happened, this was done, then this happened, and this was then done, etc.

        iii. **events:** key occurrences, e.g., that happened right before, during, or after a behavior and that seem to trigger behavior, affect it, or be its result

        iv. **objects**: stuff that is important to the story)

        v. **physical processes** (if important., e.g., weather). Note human-processes appear as behavior and behavior trajectories.

        vi. **resource dependence**: anything you find that tells who uses, produces, consumes, transfers, owns, or has some other relationship with an object

        vii. **communication**: originator, topic, and intended audience

        viii. **roles**: **a** role is a collection of behaviors or tasks that various actors may play in a similar way (e.g., doctor, lawyer, president, policeman). Titles, and anything that signifies authority goes here. Often you will see roles in role relations, e.g., doctor/patient.

3. Once you have completed the markup task, build EBGs in ACCORD for each news story

4. Describe a real choice problem -- an important decision you know about (perhaps because its yours!)

2.2.4 Client-Centered Mapping

1. Begin gathering data on a JFACC.

    a. To the extent that you can, start by gathering information on **primary** functions accomplished by the primary actors of the organization. I.e., try to look at what various groups and a (perhaps) few key individuals accomplish first, rather than trying to understand any particular job station in depth.

   b. We will proceed by refining the description after we have the overall view developed.

2. Do not start with interviews. Begin by gathering existing materials on the design and operation of the JFACC.

   a. Also look for material that expresses criticism or thoughts on how it could be better.

   b. If you happen to attend interviews, capture the results in terms of the maps that follow.

3. Analyze the materials you gather to build up the following maps. Let me know any time you find you have info you think important and you don't know how to enter it into one of these maps.

   a. **Actor // Task Map.** Your product is a map of individuals or groups, and for each, their typical tasks/jobs. The following are kinds of things we would like to know about every task/actor relationship. You won't know this about every task, but ideally, we'll get this, or most of it, for key tasks.

      i. Develop a list of all key tasks accomplished in a JFACC. How do we recognize a task?

         1. A task is basically a statement of an objective: create this product, form this prediction, make this decision, achieve this condition, certify this conclusion, etc.

         2. Sometimes you will learn of tasks directly – here's a task, and these folks over here accomplish it.

         3. Other times, you'll identify the individual or group actor, and then seek to know the jobs/tasks they are responsible for.

         4. Still other times, you will identify a database or kind of result (e.g., plan or plan update) whose contents are the result of tasks to fill or update it, and whose users evidently have still others tasks that require use of the contents of the database. So you'll infer from data types or other knowledge structure to producers (and their tasks) and consumers (and their tasks).

      ii. What person *or group* does them?

      iii. Who assigns the task, or how is it discovered/articulated?

      iv. For tasks accomplished by groups, is there authority over the process of doing the task?

        1. e.g., is someone in charge of accomplishing it?
        2. e.g., who certifies that the task has been accomplished, or should be suspended or aborted?

    v. What is the essential training the person or group has to do the work?

    vi. What counts as success or great quality for the task? Take care to distinguish deadline requirements from other measures of quality.

    vii. What counts as failure or low quality? Again, distinguish deadlines from other measures of quality…

    viii. Task data will not all arrive in the same format; you'll need to be flexible about this.

  b. **Task // Info&Choice Content Map.** Your product is the specification of key information, choices, and critiques associated with specifying or accomplishing tasks. For each task/job, identify:

    i. key analyses performed (i.e., information products created). ***be especially alert for any mention of CREATION of forecasts or predictions***

    ii. key information used

    iii. source of key information: creation, communication with others, database access, model or simulation study, …

    iv. distribution of information: who it is shared with, and how (e.g., conversation, in database that another accesses, etc.)

    v. decisions that are formed and taken, or revised/updated, as part of a task

    vi. dependence of decisions on information products (e.g., who consumes information produced by the task, and for what choices?) ***be especially alert for any mention of USE of forecasts or predictions***

4. On interviewing staff to gather information:

  a. Do not seek to interview anyone until we have done all we can without that, or at least, until we discuss the benefits and costs of doing this.

  b. If you participate in interviews then capture the information in the prior format. Ideally, tape the interviews and come back and analyze it into these two maps.

  c. For each task/job, include annotations for any critical remarks -- ways things could or should be done better, problems or difficulties, puzzles, etc. ***absolutely capture this information -- critique is often of vital importance***

  d. For what its worth, it is my hope that you and the persons you interview will find this a useful format for these interviews. If so, I can help a bit to refine it so that you can lead others to tell you information in this form.

2.3 Sample Training Interaction

In Practice Mapping training, the goal is to lead the trainee to see human systems in terms of Generative Practice Theory – to see behavior and joint behavior as the product of criteria-seeking (adaptive) generators. And, we aim to teach a form of best-practices for the design of the ontology that guides observation, event, and behavior capture.

The following is an (un-edited) example of a sample problem, mapping a newswire report, the work of the trainee, and **the commentary provided on the trainee's work**. We wish to draw attention to the way we treat the work of the trainee as an entry point into the necessary theory and a better methodology – mappers are developed through practice. In this way trainees have concrete data on which to base their learning.

Here, we show the first step in the process of becoming a mapper – learning to properly recognize and classify the elements of some sample data, shown here.

> *Source: Radio Afghanistan*
>
> *30 December 2004*
>
> *Police seize 1,485 kg of raw hashish in southeastern Afghan province*
>
> *Police seized 1,485 kg of raw hashish in [southeastern] Paktia Province yesterday. According to another report, farmers have voluntarily cleared 20 hectares of land of opium poppies in 11 villages in Chamkani District in Paktia.*
>
> *[Correspondent in Pashto] In line with the fight against the cultivation, production and trafficking of drugs, officers of the Criminal Department of Paktia Province seized 1,050 kg of raw hashish in Levan Village in the Machalghu area and officers of the Antiterrorism Department discovered 435 kg of raw hashish in Sayd Karam District yesterday. Minister Adviser and Paktia Governor Asadollah Wafa, Security Commander Maj-Gen Soleman Khel and the heads of government offices were present when the hashish was transferred to the Security Command. Mr Wafa thanked the vigilant officers of the Security Command for their fight against drugs.*

> *According to another report, farmers have voluntarily cleared 20 hectares of land of opium poppies in the villages of Star, Babu Khel, Alem, Mirjani, Hesarak and Kodgi Hesarak. Mohammad Darwesh, Paktia Province, for Radio Afghanistan.*

The trainee's map follows, **with our instructional remarks interspersed.**

Observation

- Observation Type:
    - Anti-Drug Operation
        - Start Date/Time:
            - 12/29/04, Datum Date 12/30/04
        - End Date/Time:
            - 12/29/04, Datum Date 12/30/04
        - Location:
            - Paktia Province [southeastern Afghanistan]
        - Source:
            - Mohammad Darwesh
        - Source context or role:
            - Radio Correspondent
        - Transmitting Source:
            - Radio Afghanistan
        - Contents:
            - The full data is referenced here

The time of the observation is 12/30/04. We distinguish between the time of the observation, which often reports on earlier events or behaviors, and the time of the events and behaviors.

This is an example of the way observations are evidence for *inferred* events and behaviors…

Note that events and behaviors therefore each have underlying observations – in this case, each event and behavior has as evidence an extract from the source. In turn, the extract points to the overall document.

Since you'd have many observations if you did things this way it's probably better to start using ACCORD…

Observation context generally means the conditions that led to the observation being made. We don't really care here.

Actors

- Drug Smugglers
- Officers (Criminal Department of Paktia Province, Antiterrorism Department)
- Officials (Governor, Security Command Members, and Government Offices)
- Farmers

Lumping actors into super-actors, as you've done, is useful in some cases but not all. It depends on whether we'll have an interest in the constituents as we continue to model. Here, we'll want to break them out (see notes on Behavior, below).

Security Command is an important actor. We may suspect that this actor has as constituent actors the Criminal Dept. and the Antiterrorism Dept. In a typical mapping engagement, we would confirm or disconfirm this, by establishing the relationship among the three actors.

Events

- Event Type: Drug Seizure
    - Location:
        - Paktia Province: Levan Village(Machalghu area), Sayd Karam District
    - Drugs and amounts seized:
        - Hashish: 1,050kg Levan Village; 435kg Sayd Karam District

If we were doing this in the tools, you'd distinguish the drugs property/feature of the event type "drug seizure" from the amounts.

I have a preference for more general event types, i.e., "seizure" over "drugs seizure." The generality of the type is always important to consider – i.e., you are making an important choice here. In this domain, arms are seized too. So to create the "drug seizure" type is to map it as if this is the only seizure we care about, or to commit to separate types for separate kinds of items seized – e.g., "drug seizure", "small-arms seizure", "wmd seizure". The important thing is to think about the practice of the actor. If seizing one thing versus another is essentially the same behavior, then the event likely abstracts over the various things seized.

- Event Type: Land Clearing
    - Location:
        - Paktia Province: Chamkani District(11 villages)
    - Drug type and land size:
        - Opium poppies: 20 hectares of land

You evidently chose this type from the text of the observation. But if you read the text again, you can see that it's not about land, but rather, poppy eradication… "clear the land of opium poppies"… That is what the text is really conveying, and it is an event pertinent to the practice of interest. So the event type "eradication" would be more suitable.

Again, drug type and land size are separate features of the type.

Behaviors

- Behavior Type: Anti-drug Operation
    - Location:
        - Paktia Province: Levan Village(Machalghu area), Sayd Karam District
    - Actor:
        - Officers
    - Location:
        - Paktia Province: Chamkani District(11 villages)
    - Actor:
        - Farmers

Earlier, the event type was (perhaps) too specific. Here, the behavior type is (perhaps) too general. There are lots of anti-drug operations. We see this because there are four behaviors, not one:

- **seizure (implying a planned raid),**
- **discovery (implying they were doing something else not mentioned)**
- **transfer to the Security Command**
- **eradication efforts**

Each should be separately mapped, because they are distinct occurrences, and because they are accomplished by distinct actors (and so evidence for separate practices!!!!). Always keep in mind we seek to understand practices – event and behavior mapping is in service to this primary aim.

We'd like other info to decide best how to map the accidental discovery. We can safely assume the officers are always looking for drugs, whatever they are doing. Knowing only what we know, I'd say we should focus on a single behavior type, "SeizeContraband", for the seizure and the discovery, since the latter does involve seizure… Later, if we learn of

other practices that may (as a side-effect) uncover drugs, we can model this. I would strongly oppose a "discover" behavior.

Note the discovering actors are in the Antiterrorism Dept. In some mapping engagements, we would treat this as a clue to their practice. Here, I think the key inference to draw is that fighting terror and fighting drugs are correlated. Otherwise, the anti-terror guys would just hand off the drugs to the criminal dept and go back to their stuff, whereas here, they both are named, both get credit, etc.

Objects

- Hashish
- Opium poppies [land]

Land is separate

Resources

- No data

Depends on what we want to model. The land, for example, is a resource for farmers. And later, the poppy plants function as a resource for the exchange practices that sell them for money…

Communications

- No data

Again, depends on our modeling interests. The statement by Mr. Wafa may be worthy of note – e.g., if we are modeling the emergence of anti-drug rhetoric in the nation.

In general, the comments of participants on activities are VERY instructive.

Here, you should map it just to think through the communication activity.

Roles

- Drug Smuggler
    - Cultivate and Manufacture Drugs
    - Procure, hide, and smuggle drugs out of country
- Drug Control
    - Investigate and probe for drug smuggling activity
    - Seize drugs
- Farmers
    - Voluntarily assist in the reduction of drug production

We do not actually know much about the contents of these roles from this one document, especially for the smugglers and officers.

Roles are three things:

1. **a set of tasks that the role-playing actor nominally undertakes; or, dually, expectations for kinds of events (usually, over time) that will be produced**
2. **a collection of choices that are customarily part of accomplishing the tasks**
3. **resources customarily or necessarily available to accomplish the tasks**

We don't have a lot of info in this document, but, you should go back through and think about the roles along these three dimensions. You should do this with all other role-analyses going forward.

One reason role analysis is important is that we have few of the actual participant actors – we are learning about kinds of behaviors/practices rather than about the behavior of specific individuals. Equivalently, we have actor types but not actor instances for the

farmers and officers. So the actual actors are things like Farmers_instance#, CDPP-Officers_instance#, etc. It is typical for this work that we learn about kinds of behavior without knowing more than the kind of actor – and so we focus on roles.

You might think about this distinction from the point of view of simulation. Here, we'd spawn actor instances as needed.

Clearing land of poppies is much less than assisting in reduction of drug production. They can replant next week… We have to stick with what they actually did, in the concrete circumstances that obtained.

Critique

- There is this 'According to another report' reference which I don't know how to markup. It almost seems like an actor but not modeled such as the correspondent. I wouldn't call it a resource because it didn't affect behavior. I wouldn't call it an object because it's not an item in an event. I decided on not mapping it because it seems like nothing more than a conduit for information such as the correspondent.
    - **Fine. If we capture the observations properly we get this for free.**
- The final paragraph repeated the Land Clearing so it didn't need to be mapped again but it did contain a little more information; several village names.
    - **The inferred events and behaviors have as evidence information from various patches of text (collected as one observation).**
- For the Land Clearing behavior, I wasn't sure if it was right to lump all the villages under Paktia Province and/or Chamkani District or to list as many villages as possible or just the number involved. I went with Paktia Province: Chamkani District(11 villages). I figure the specific village names can be looked up later on if they are needed.
    - **As I said above, these are to be separately mapped. They come together via the transfer behavior.**
- There was no communication to markup, except between Wafa and Security Command which isn't of special interest.
    - **As per previous comments, we do not know this. It could be the most important thing to map.**
- I found no data on resources that effect behavior.
    - **I'd err on the side of trying to assign every object as a resource for some practice.**

    - **If you – as analyst – are not going to practices yet (e.g., you are just doing the EBGs) then a solution to keep things around is to create a feature of a behavior type such as “associated resources”**

- For the physical date I used the day before the date listed on the datum because the events took place ‘yesterday’ according to the report. Since we are capturing the observation itself I also put down the date on the datum.
    - **Events and behaviors are ‘yesterday’, observation is ‘today’.**

- I joined the Criminal Department of Paktia Province and the Antiterrorism Department together as one actor called Officers because they filled the same role and performed a similar event. I did decide to make note of this join in the Actors section. Same goes for the Officials group.
    - **This decision needs to be made in terms of the actors or practices we seek to understand – i.e., not because they have a prima-facie similarity of roles.**
    - **When in doubt, don’t combine.**

- I made note of the Officials for future reference but there wasn’t any reason to map them here.
    - **Since the transfer and communications are important, you need them.**

Appendix A: Generative Practice Theory

Generative Practice Theory – Modeling Behavior, Skill, Individuals, Organizations, and Culture

We describe our theory of human systems, **Generative Practice Theory.** This theory contributes a generative mechanism sufficient to model primitive and complex skills, individuals and their personalities, and any desired social formation, such as teams, organizations, and institutions. We begin with the conceptual foundations, than present our theory of personality, ideology, and culture as extensions to the base theory.

Conceptual Framework

Situations never exactly repeat. And, there is no behavior that is an exact image of another prior behavior. Instead, human individual and social competence creates similar, valued outcome situations, for the widest possible range of input situations. This ability to adapt to a variety of situations, and to regularly produce more-favored situations, reveals the presence of criteria. Criteria name the values-in-action, the 'control laws' that govern or guide – adapt – behavior. The presence of criteria establishes how behavior can be provided for and managed within situations *the actor has not yet met.* Intelligent, resource-constrained actors must have this capability – lacking omnipotence, we must be capable of adapting to new situations. Criteria-managed behavior is adaptive behavior.

Representing Behavior

We distinguish **behavior**, which is a manifestation of an actor's skill, from practices, which formalize the skill as a whole. **Practices** are hypothesized generators sufficient to produce and adapt behavior as a function of situational conditions. **Situations** are collections of events. Behavior is what we observe, while practices depict the capacity of the actor to produce and adapt – that is, manage – behavior. From the point of view of analysis, behaviors are evidence for practices, and practices hypothesize the generator that produces those (and other) behaviors.

We have developed and validated a representation language to capture observations about behavior. This language, **Event-Behavior Graphs (EBGs)**, is a directed graph formed from event and behavior nodes, and directed links. Event nodes depict occurrences and relations over occurrences, and have an associated temporal extent (an interval). Events may depict physical, cognitive, and social states of affairs. Behaviors are activities that abstract over a set of events, have a temporal extent, and are identified as being undertaken by some actor. The actor may be an individual or a social actor.

A link from an event into an event represents temporal precedence. A link from an event into a behavior signifies that the event is either a (partial) cause that triggers the event, or an element of context (like the location) that shapes the way the behavior transpires. Links from behaviors into events signify that the events were produced by the actor via that behavior. Links into and out of behavior are hypotheses (of an observer) about the intentions and performance capabilities of the actor producing the behavior – as per the previous remarks on criteria-driven behavior.

Event-behavior graphs are hierarchical because events can summarize entire event-behavior graphs, and because behaviors can summarize over constituent behaviors. In the latter case, if the upper-level (abstracted) behavior is conducted by an individual, then the refinement nodes depict sub-methods of the individual, and their intermediate products and results. If the abstracted behavior is the product of a social actor, then the refinement graph depicts constituent contributions of actors that are themselves constituents of the social actor that produces the abstracted behavior. Hierarchy in EBGs enables us to model at any desired level of abstraction, and then to refine elements – componential and social decomposition – as needed.

At the end of this document we include a brief image of Synergia technology for developing and managing event-behavior graphs.

Practices – Modeling the Generators of Adaptive Behavior

We now describe the fundamental construct for our work, the practice. Then we show it is sufficient to represent individual and social actors, and to enable us to formalize what we mean by personality, ideology, and culture.

The language of practices enables users to formalize observations of situated behavior in terms of a summarizing dynamic capacity. A **practice** is defined by a triple: a set of **situations S,** an **action generator A,** and a set of **criteria C.** The situation represents the information state monitored and created by the practice, as behavior is produced by the action generator. Criteria evaluate the current situation and orient behavior to improve it, and may take the form of goals, proscriptions, preferences and tradeoffs, conditions that legislate among contingencies, success and failure conditions, interrupt conditions, etc. We can use a propositional form to express the relations among its elements:

For all Situations of type S, Do Actions A so as to enforce Criteria C

Practices are behavior generators that seek to satisfy some criteria by transforming antecedent situations into more satisfactory consequent situations. Actors regularly seek to create particular kinds of consequences, often for a very wide variety of starting conditions. Just as conditions vary, behavior never exactly repeats in all of its details. Of course, it is impossible for actors to embody a separate rule or behavior for each of the infinite number of feasible starting conditions. It is impossible to learn all the distinct rules, and impossible to store them. Thus we are led to the central innovation and importance of the practice construct. By introducing criteria as a foundational component of behavior production, practices make it possible for actors to embody a very compact 'machine' that adaptively manages behavior production across a range of events. Criteria enable actors (and us as modelers!) to 'fill in the gaps' and so generate effective behavior in new situations.

In general, A may be adjusted multiple times, even continuously, to (attempt to) satisfy C. This leads us to express the above propositional form as a dynamic system, or equally, as a computational system. Practices, as systems, are composed of a set of monitors, a set of action generators, and a set of data managers. For this form we introduce execution resources, R, since behavior is the product of a physically-embodied actor and so necessarily consumes resources during execution. We have:

```
Concurrently, subject to resources R:
    Maintain a description of the situation S ;
    Monitor for departure from the satisfaction of criteria C in S ;
    Supervise execution of action generator A to maximize satisfaction of C in S ;
```

Monitors are responsible for activating the generator A, and for suspending it under conditions of success, failure, resource limitations, or as directed by practices that contain this one. A is designed to be advisable by the deviations measured by the monitors – action generator inputs include the primary information relevant to the behavior, as well as an 'error signal' created by the monitors. Monitors determine the ongoing state of resources – a part of the situation for the practice – and deposit signals that manage A with respect to R, or undertake actions to acquire additional resources. Data managers maintain the situation S. They accept data generated external to the practice and maintain it as well as internally-generated data. They may keep all or part of the data, and may manipulate the data to best serve monitoring.

We develop models of actor practices from the information in event-behavior graphs. The graphs record situations – collections of starting, in-process, and concluding events. They also depict the underlying action logic and so the program that defines the action generator – the sequential, concurrent, and conditional relations over behaviors that intervene between antecedent events S^- and consequent events S^+. The action generator responds to the situation, to better satisfy the criteria. We analyze the transformation from S^- to S^+ within each EBG, and across EBGs – variations in the situations and action logic extend and refine our understanding of the relevant situations and actional capacity of the actor – and above all else, the criteria rationalize the transformation.[5] For example, consider behavior variation (for some practice) that is attributable to resource variation. Here, we may analyze the behavior for what is preserved versus left aside, to see how the actor exploits (or copes with) differing qualities, quantities, or availability of the resource. We learn an adaptation scheme that embodies (nothing need be articulated!) a tradeoff that favors some situations/results over others, as a function of resources. Thus a practice is a generator that strives to transform a set of situations into some other set deemed more favorable (as judged by the practice's criteria).

Generative Practice Theory embodies a theory of mimetic learning as a basis for the formation of new practices. Mimesis is a process of observing manifested behavior, then building a practice that can produce a semblance of what is observed, using ingredient practices and practice-elements already in the actor's repertoire. Note that the practice is the generator of behavior – and so actors cannot observe the practices of others per se. Rather, we see the overt manifestation of the execution of the generator (i.e., we see behavior, not the generative competence). Therefore building a practice is a kind of reverse-engineering that begins with a copy of some behavior, and draws on the existing practices of the actor to compose one that reproduces the observation. This process tends to produce an imperfect or at least divergent capacity than the one observed, at least initially. Over time, the social system will tend to guide the actor, rewarding and punishing, guiding, and exhibiting behavior that is used to develop new observations, so that the practice is refined. Individuals and organizations employ the same mechanism.

Practices may contain other practices. For example, the action generator A may itself be a set of embedded practices that execute concurrently, or may be a program that imposes some degree of sequential operation over the embedded practices. *This means we can model individuals, organizations, and other social forms as systems of practices.* A rereading of the propositional and system forms shows that they make no commitment to whether the practice that is described

[5] Since criteria are so important to this work, we offer three remarks on their development. First, criteria can be depicted as a partial order over situations - consequents in the situation set S^+ are preferred to prior situations, to still-prior situations, and so on until antecedents in the situation set S^- are reached. (If progress is not uniform, then we are afforded multiple opportunities to observe the preference in action.) Quantitative relations are derived by consideration of quantitative features of the situations in this ordinal ranking.

Second, if we can learn what the actor monitors – events that signal the degree of satisfaction of the criteria – then we have useful evidence for the property that is being tested, and so, of embodied criteria. It is in the nature of human value-seeking behavior that we aim to increase, decrease, or preserve some quantity or state of affairs in our actions. Therefore, measurements of an item can be evaluated for the purpose of increasing, decreasing, or preserving that item, its source, or its projected consequences.

Third, we have developed applications of Generative Practice Theory to formalize personality, ideology, and culture. Each of these entails a discipline for analyzing variation in EBGs to identify embodied criteria. Of these, the criteria embodied by the culture are generally accessible and are extremely influential for the actors that participate in the culture. In other words, we have access to very useful information about an actor's criteria, just by knowing the culture that the actor of interest (individual or social) inhabits.

represents a skill, the overall behavior-generating capacity of an individual, or the capacity of some other social form. In short, the practice construct is sufficient to represent individual and social behavior!

Practice Inter-dependence – Individual and Social Actors

Actors – individuals, teams, organizations, institutions – are bundles of inter-dependent practices. Let us look first at the fine structure of inter-dependence. Consider two practices, Π_1 and Π_2, where Π_1 is defined by the triple (S_1, A_1, C_1), and Π_2 by (S_2, A_2, C_2). S_i is the set of all situations that the criteria C_i might dictate a response to. As a result of activation and supervision by C_i, A_i creates some of these situations. Practices are inter-dependent whenever the situations S_i of interest to one practice Π_i are at least partly created by some other practice Π_j, resident in the same or some other actor. Ultimately, inter-dependence reflects a coupling among the criteria that activate and manage practices – since criteria manage action generators to create satisfactory situations. For example, if part of S_i is created by Π_j, this means that the criteria C_i respond to and seek to modify situations that have been promoted by C_j – inducing the criterial relation between C_j and C_i, through S_i (via A_i).

The inter-dependence of practices is not always realized in behavior, much less observed. Practices are coupled if their criteria are *potentially* coupled – that is, coupled via situations that *might feasibly be* created by the action generators (and in light of ongoing physical processes). Therefore, actual inter-dependence of practices far exceeds whatever has been observed.

The existence of the many potential interactions among practices (and so, actors) gives rise to a type of field, a **practice field,** among practices/actors. Just as for a electromagnetic field for any set of coupled practices, the specification of conditions for a subset of those practices constrains the behavior of all others and so the field as a whole. And again, just as we see in electromagnetic fields, the dynamic properties of executing practices (e.g., how fast they suspend once triggered) constrains the rate of propagation of conditions through the practice field.

We have discussed how practices influence one another. We now turn to the two large-scale types of inter-dependence found among practices, representing ways that interactions among practices have stabilized for some period of time. One defines the cognitive structure of individual humans, the other, relations among the practices of actors in an organization or other social formation.

Individuals are systems of practices that are prioritized and coordinated within a supervisory actor architecture called the **Ambiguity Reduction Architecture** – the 'operating system' of the individual. The architecture treats the recurring problem individuals have: what to do next. Ambiguity reduction abstracts the large body of empirical results on strategies individuals use to focus and re-focus attention, resolve conflicts, select among optional actions, manage interrupts, suspend and re-activate cognitive and other tasks, etc. in three ways. First, there are structural relations over practices: containment or nesting relations, mandated sequences, blocking rules, etc. Second, for those practices that may jointly seek to execute, there are control relations/policies that prioritize and schedule or suspend their execution. As needed, the actor may deliberate to develop for fully specialize a policy for current conditions. Third, policies determine the portion of available execution resources to distribute to the various practices that are scheduled to execute.

Control policies in this architecture may or may not be ‘knowledge based’ – actors are expected to make use of content-specific control or choice-relevant information if available, but ultimately, may legislate which practices get execution resources based on general rules (e.g., flip an n-sided coin). However, all control policies are realized by making execution resources available to some practices (e.g., execution resources may be provided to any subset, or to a program that partially or totally orders a subset of the practices).

Organizations, and other social forms such as teams and institutions, are patterns of regular inter-dependencies among practices that, in turn, are ‘carried’ by individual and social (multi-individual) actors. We distinguish three types of inter-dependencies among practices – three ways that behavior enacted by some actors may influence the behavior of other actors. These are resource relations, communication relations, and role relations.

The **resource** relationship among practices establishes the way actors inevitably interact with respect to physical ingredients – physical objects, physical settings, and information. Resource inter-dependencies depict how practices *jointly influence* one another as they create, modify, share, transport, consume, and otherwise manipulate resources (material, information).

The structure and flow of resources co-stabilizes with the practices that appear in the resource relation. Criteria dictate distribution, though perhaps not uniquely. However, for stable situations, and for critical resources, one to a few stable practice fields may emerge, and once having done so, stabilize – so that other resource relations, and other more general practice relations, adapt under the assumption of the resource relation as it stands.

The **communication** relationship establishes the flow of interpreted information through actors. Communication inter-dependencies are tendencies to focus on, manipulate/interpret, seek, share, and be advised by kinds of information in given situations, with given actors, as parts of ongoing activities. A communication relation is present provided multiple actors are capable (in their practices) of initiating, receiving, and making use of some information, and provided they share a tendency *to actually do so* in some collection of situations.

Therefore, a communication relation is not present simply because two actors happen to be connected via some communication medium like a phone or e-mail link – this is actually a resource, and may figure in a resource relation. A communication relation is a relation among practices. A necessary but not sufficient condition for the presence of a communication relation among a set of actors is that they are all capable of forming, sharing, and being advised by some topic. It is also necessary that the actors are actually disposed to share the information with others, for some collection of situations that can actually arise. Together, these conditions underwrite a communication relation.

As with resources, communication relations tend to stabilize in the organization – once criteria finds ways of satisfying themselves, their action generators tend to be used in like situations, and defended until blocked or otherwise found unsatisfactory. For example, we tend to come to rely on particular actors for particular kinds of information, individuals tend to develop idiolects and groups tend to develop dialects, we are willing to share information or engage in kinds of communication with one sort of person but not another, etc.

We remark that resource and communications *networks* are a minimal abstraction of the practice relation we describe. They depict the existence of an influence between multiple actors, but suppress the space of potential responses one actor might make as a function of an element in the space of actions an influencing actor might take.[6] E.g., the many responses one actor might make to others consuming resources, or the variable interpretations and responses an actor might develop due to the particular pattern of information present and absent, and its quality.

Role relationships establish the orientation actors have with respect to one another. A **role** is a systematic response an actor makes to a class of situations created by current context and, especially, events created by actors in coupled roles (coordinated or conflicting). Therefore, roles are members of role relations. Roles embody established kinds of interpretation, including expectations for other actors and for likely outcomes, the division and coordination of labor or other activity, and the distribution of authority and responsibility in a social system. Role relations (e.g., patient/physician, leader/follower, adversary/adversary) reside in the way participants tend to contribute to a joint activity in ways that enable the participants as a whole to jointly move things forward. They make effective action feasible among relative strangers (once situations relevant to the role relation are brought into being, and recognized by one or more participants).

[6] Practice inter-dependence is a many-many relation – each practice may influence multiple others, and multiple practices may influence one practice. Further, the product of a practice is not always certain, even given its inputs – and at a sufficiently fine level of description, the enactment will entail fundamental uncertainties. Therefore, each practice induces a space of possible responses, for each of its admissible influencing states. In turn, admissible input states are generated by the practice, other practices of the actor, and practices of other actors (interacting in the physical environment).

Roles are summaries over the practices of actors that embody the role. For example, our understanding of the role “physician” derives from our abstracting over the behavior of many individuals acting as health-care providers. It is useful to conceive of a role as a task structure. Tasks reflect a belief system, objectives relevant to various situations, legitimate ways to undertake the task, and relations to other tasks. An actors’ practices instantiate particular skills for recognizing relevant situations, forming relevant interpretations, setting and satisfying objectives, etc.

Roles and role relations structure communication and resource relations. Since most behavior is to a greater or lesser degree social behavior (behavior that is influenced by, or influential for, other actors), most behavior is structured by roles. Communication and resource utilization are in service to the criteria of the actor. The active criteria are determined by the currently-active practices, and so, by the current roles. For example, topics of communication, and physical settings and ingredients, are oriented by the role – departures from the role-based norms are confusing, seen as wrong or inappropriate, or in a few cases, humorous.

A role is realized within an actor as a system of practices that regularly interacts with other actors’ practices that embody the complementary role(s). Therefore the criteria embodied in the several roles must be coupled – moving things forward for some or all participants, or, perhaps, moving things forward for some participants, while thwarting others’ ability to satisfy their criteria (adversarial roles). Criteria (and so their practices) may be coupled over outcomes – events created by some practices are ingredient input situations for other practices. Also, criteria may span the joint activity – as when the valued outcome is a group or system outcome (e.g., teams, not individuals, win the soccer match). In the first case, participant roles guide participant behavior in light of information about self/local actions, and guide interaction at the boundaries of the practices. In the second case, participant roles guide participation by measuring global/shared outcomes, and advising individual contributions to serve the creation of the global outcome. Both kinds of criteria, and attendant action generators, are generally present in a role.

As things proceed, a role relation achieves its central objective and also maintains the relation itself – stabilizing the practice relation, just as recurrent resource-sharing and communication acts stabilize those practice relations. For example, the patient and the physician jointly act to produce the diagnosis, treatment, and healing, and the way they do so preserves their constituent roles. In a care episode, the criteria that lead to healing are satisfied in part by observing the role relation – and so the roles are sustained because (in relation) they are generally effective.

Some roles are taught. For example, physicians are taught the physician/patient relation and its tasks, objectives and belief system, while patients and physicians experience it repeatedly and, as new participants, are guided by the responses of others (novelty and error are limited). Other relations are ‘absorbed’ by the stabilization of relations in activity, rather than being articulated and taught – consider the roles siblings have with one another and their parents. In formal organizations (government, business, societies, etc.) there is some of each. In all cases, the various parties to the role relation create a pattern of initiatives and responses that lead new participants to conform to a (quasi-) stable relation.

For each of resource, communication, and role inter-dependencies, the relationships among participant practices determine the kinds of interactions that are possible, and as part of this, possible products that are shared and shareable. Therefore, “organization” is not what passes between actors – e.g., the physical resource, or the communication topic. Rather, organization rests with situation/action/criteria complexes that co-evolve, stabilize around, and ultimately determine recurring types of joint actions. This is easiest to see with role relations – there is not necessarily any ‘thing’ that passes between participants. In short, ours is a psychological and social view of organization as a structured, quasi-stable deployment of adaptive cognitive competence, that is, of practices.

Foundations

We cannot derive the practice construct from first principles. Instead, we can show that what we know limits the range of possibilities for what can count as a generator of human (individual and social) behavior. We develop the construct from this point of view.

Pertinent Empirical Facts

- Every behavior is unique
- Cognition is bounded = we are finite
  - We are not capable of infinite discrimination, nor of perfect control
- Likenesses exist among events and event-relations
  - And so, among situations and behaviors (which are event complexes, e.g., an event set, or event trajectory)
  - We generally cannot produce a final definition of, say, a behavior, but we generally can say whether one behavior is or is not sufficiently similar to another
- For a given actor and epoch (window in time), we may observe an actor *managing* a behavior in a regular manner – provided the epoch is long enough to observe multiple manifestations. In particular:
  - Similar starting situations often precede similar behaviors, that in turn produce similar results
  - For a given behavior, we often observe a wide range of antecedent situations is controlled, as a result of behavior, to a tight range of consequent situations.

        - For longer epochs we may observe evolution of the behavior. E.g., we may observe the range of starting conditions growing and/or the variation in the consequents being reduced – i.e., increased competence.
    - Path-dependence is pervasive.
        - An actor's milieu is dynamic, and the entry points for behavior operate over a period of time – rather than appearing at a single instant, only.
        - To the extent that the nature of the behavior permits it, actors can begin or re-enter behavior at multiple starting points.
    - We observe both homeostatic and transformational/goal-oriented behavior
- As actors interact, behaviors manage other behaviors
    - E.g., plan execution by individual and social actors
    - E.g., competition, coexistence, collaboration
- Except for our biological endowment, the management of our behavior appears to be learned through interaction with the physical environment, with ourselves, and with our social environment.
    - The most basic mechanisms seem to derive from our ability to reflect – to produce an image of – the behavior of others. Equivalently, mimesis is the foundational learning mechanism.
    - E.g., we become ourselves, and adapt to fill a variety of social roles, by conforming to all these others – to mirror/mimic, complement, or oppose them.

Developing the Practice Construct

- We postulate a generator for the behavior – a capacity, embodied by (within) each actor that produces the behavior
- The set of potential starting situations (for any behavior) is infinite. Yet all actors are finite.
    - Therefore the generator is finite, and its discrimination is finite.
    - Therefore the generator cannot be a pre-existing map from antecedent situations to an open-loop/reflex behavior
    - Therefore the generator must have the form of a capacity respond to a range of antecedent situations, including never-before-seen ones
- We identify the regular management of antecedents into consequents (i.e. similar events leading to similar behaviors, producing similar consequent events) with the values of the actor.

    - Equivalently, behavior inevitably expresses value, and so a value construct needs to be associated with every level of behavior production
    - Values are relations over situations.
        - Recall that the actor experiences the antecedents, and evidently strives to create something in a recurring range of consequents.
    - Values may be qualitative (logical, symbolic) or metrical.
        - Minimally, a value is a partial order over situations. (Relevant situations are those that transpire during behavior, and those projected to transpire absent behavior.)
        - Often they can be identified with conditions that the actor strives to prevent, undo, decrease, maintain, increase, or create.
        - In some cases, a measure can be identified that is extremized during the course of behavior. In this case, the measure orders situations.
- A finite generator with the above properties, and sufficient to reproduce the empirical data, can be developed from three constituents – a perceptual system, a behavior-production system, and a criterial system. We call the abstract machine with these three components a practice.[7]
    - The perceptual system maintains a dynamically-changing situation, whose contents are relevant to the behavior-production and criterial systems
    - The behavior-production system is an advisable/controllable action-generation capacity
    - The criterial system monitors the contents of the perceptual system and generates signals that direct the ongoing action of the behavior-production system.
    - The three systems operate entirely concurrently (i.e., unlike the simpler state-based control system architecture – whose plant and control subsystems both tick ahead to the same clock).
- We do not presume all behavior is generated by single totalizing practice – there is no evidence for one, and substantial evidence against one. Instead, we postulate that actors are composed of many practices.
    - However, we have no need to suppose that the form of the practice needs to diverge from the three constituents just mentioned. Equivalently, any specific

[7] The construct is distinct from any particular embodiment – which is the generator of some particular behavior. If necessary, we could refer to the "practice construct" and then specific practices. Alternatively, we could retain "practice" to refer only to the construct, and then qualify the specific ones – e.g., a managing-a-meeting practice. Here, we allow "practice" to refer to either, trusting the context of discussion will clarify which usage is intended.

    behavior generation capability can be instantiated using these three constituents of the practice (construct).

- To support the fact that behavior manages other behavior, we require a) that practices can be coupled to other practices, so that the situations produced by one set can be coupled to the perceptual systems of another set, and b) that practices can be recursively embedded.
    - Recursion provides for general control structures among practices.
        - E.g., an outer practice that designs a policy in a situated manner, communicates it to an embedded practice, which installs it in the embedded criterial system, that (recursively) uses the policy to guide behavior production

Appendix B: Sample Practice Schema

We illustrate the use of the quasi-executable specification language.

We show a practice, taken from a scenario we invented – a manhunt for a key figure who we never see. Instead, we see the effect his behavior has on the physical and especially social environment (i.e., we do not observe him, but we observe the ripples created by his passage). Details on this scenario are extensive – we used it to test our mapping and modeling technology, as well as this specification language, and so a version of the scenario exists as a sample project in ACCORD.[8]

This particular practice shows a critical choice of the primary character – the "Key Figure." He must assess the threat of capture that he faces and determine his next move.

```
practice EscapeToSafeHaven
{
   /* 1. Information Environment
    *
    * Set variable names are always of PLURALS. Other variables will be of singular form
    */

   set Locations = { "Settlement A", "Settlement B", "Settlement C", "CaveComplex I",
                 "CaveComplex II", "CaveComplex III", "CaveComplex IV" };
   set Activities = { "Encamped", "In-Transit" };
   set ThreatIntensities = { "none", "low", "medium", "high" };
   set TimeToThreat = { "unknown", "days", "hours", "imminent-current" };
   set WeaponTypes = { "small arms", "SLMs", "weaponized sarin" }
   set EscapeDifficultyLevels = { "low", "medium", "high" };
```

[8] The details of this scenario are available upon request.

```
String KeyFigureResource = "small-arms, weaponized sarin, radio, cell phone, trucks, currency";

String LowQualityTroopResource = "small-arms, radio";

String MediumQualityTroopResource = "small-arms, radio, trucks";

String HighQualityTroopResource = "small-arms, RPGs, SLMs, radio, cell phone, trucks";


// This variable is used to monitor and keep track of the threat the KeyFigure is facing

ThreatIntensities currentThreatIntensity;


// Used to keep track of the current threat that KeyFigure is facing

Boolean NoThreat;

NoThreat.setValue(true);

Boolean AerialSurv;

AerialSurv.setValue(false);

Boolean PatrollingTroopsNear;

PatrollingTroopsNear.setValue(false);

Boolean StrikeForceNear;

StrikeForceNear.setValue(false);


// Used to keep track of the time until the threat occurs

TimeToThreat timeUntilThreat;


// Used to show the KeyFigure's escape difficulty level

EscapeDifficultyLevels escapeDifficulty;


Real estimatedTimeThreatContactProbability;
```

```
Object escapeDifficultyProbability { Real L, Real M, Real H };
escapeDifficultyProbability.L.setValue(0.0);
escapeDifficultyProbability.M.setValue(0.0);
escapeDifficultyProbability.H.setValue(0.0);

// These store the percentage of each type of troops that have not been sacrificed
Real remainingLowQualityTroop;
remainingLowQualityTroop.setValue(1.0);
Real remainingMediumQualityTroop;
remainingMediumQualityTroop.setValue(1.0);
Real remainingHighQualityTroop;
remainingHighQualityTroop.setValue(1.0);

// Whether the KeyFigure has successfully escaped to selected Safe Haven.
// It is used to control the monitor's execution
Boolean successfulEscapeToSafeHaven;
successfulEscapeToSafeHaven(false);



/* 2. Monitor
 */
monitor monitorThreatAndExecuteEscape
{
   monitorThreatAndExecuteEscape()
   {
      Boolean l_NoThreat = NoThreat.copy();
```

```
        Boolean l_AerialSurv = AerialSurv.copy();
        Boolean l_PatrollingTroopsNear = PatrollingTroopsNear.copy();
        Boolean l_StrikeForceNear = StrikeForceNear.copy();

        ThreatIntensities l_CurrentThreatIntensity = currentThreatIntensity.copy();
        TimeToThreat l_TimeUntilThreat = timeUntilThreat.copy();
        EscapeDifficultyLevels l_EscapeDifficulty = escapeDifficulty.copy();

        ComputeThreatIntensity compIntensity = new ComputeThreatIntensity(currenThreat,
currentThreatIntensity);
        ComputeEstTimeThreatContactProb compThreatProb = new
ComputeEstTimeThreatContactProb();
        ComputeEscapeDifficultyLevel compEscapeLevel = new
ComputeEscapeDifficultyLevel();
        LeaveTroopsBehind leaveTroops = new LeaveTroopsBehind();
    }

    void startMonitor()
    {
        while (successfulEscapeToSafeHaven == false)
        {
            par
            {
                // These two calls run concurrently
                if (l_CurrentThreat != currentThreat)
                    compIntensity.Main();
                if (l_CurrentThreatIntensity != currentThreatIntensity)
                    compThreatProb.Main();
            }
            // Successive calls
            if (l_TimeUntilThreat != timeUntilThreat)
                compEscapeLevel.Main();
            if (l_EscapeDifficulty != escapeDifficulty)
                leaveTroops.Main();
        }
        ExitToSafeHaven(compIntensity, compThreatProb, compEscapeLevel, leaveTroops);
    }
}

void Relocate()
{
    // Does the relocation of KeyFigure and troops
}

/* ***
 * This is a stand-alone procedure to illustrate procedure construction.
 * It body can be embedded in the above monitor as well
 */
void ExitToSafeHaven(ComputeThreatIntensity compIntensity,
ComputeEstTimeThreatContactProb compThreatProb,
            ComputeEscapeDifficultyLevel compEscapeLevel, LeaveTroopsBehind
leaveTroops)
{
    // Does any appropriate clean-up here
    compIntensity.Kill();
    compThreatProb.Kill();
    compEscapeLevel.Kill();
    leaveTroops.Kill();
```

```
        // And additional clean-up here if needed.
    }




    /* 3. Activity Generator
     */
    activityGen ComputeThreatIntensity
    {
        // Local reference of parameter list
        ThreatTypes theCurrentThreat;
        ThreatIntensities theCurrentThreatIntensity;



        ComputeThreatIntensity(ThreatTypes currentThreat, ThreatIntensities
currentThreatIntensity)
        {
            theCurrentThreat = currentThreat;
            theCurrentThreatIntensity = currentThreatIntensity;
        }


        // This activity generator continuously monitors and modifies the currentThreatIntensity
"directly"
        // There is no need for a local copy of the parameter list
        //
        // If but there the need for a local copy of the parameter list, then the function Initialize()
        // kicks in to initialize the value variable(s), using copy()
        void Initialize(ThreatTypes currentThreat, ThreatIntensities currentThreatIntensity)
```

```
{
   // localCurrentThreat = currentThreat.copy();
   // localCurrentThreatIntensity = currentThreatIntensity.copy();
}
void Main()
{
   if (StrikeForceNear || (AerialSurv && PatrollingTroopsNear))
      theCurrentThreatIntensity.setValue("high");
   else if (AerialSurv || NoThreat)
      theCurrentThreatIntensity.setValue("low");
   else if (PatrollingTroopsNear)
      theCurrentThreatIntensity.setValue("medium");
}
void Pause()
{
}
void Resume()
{
}
void Kill()
{
   theCurrentThreat = null;
   theCurrentThreatIntensity = null;
   isKilled = true;
   /* The following may be a self-generated call. User does not need to define it. If the
assumption
    * is correct, it would the call to the kill() method of this thread object.
    */
```

```
      dispose();
    }
  }


  // Since this activity generator does not behave like a continuation, we can safely ignore the
local copy
  // of the parameter list. Also, since the generator modifies the triggering event(s) directly, we
can
  // completely manipulate them through the use of globals.
  activityGen ComputeEstTimeThreatContactProb()
  {


    void Initialize()
    {
    }
    void Main()
    {
      Real totalProb;
      totalProb.setValue(1.0);


      if (No Threat)
      {
        if ( timeUntilThreat == "days")
          totalProb.setValue(0.8 * totalProb);
        else if (timeUntilThreat == "hours")
          totalProb.setValue(0.2 * totalProb);
        else if (timeUntilThreat == "imminent-current")
          totalProb.setValue(0.0);
      }
```

```
if (AerialSurv)
{
   if ( timeUntilThreat == "days")
      totalProb.setValue(0.6 * totalProb);
   else if (timeUntilThreat == "hours")
      totalProb.setValue(0.4 * totalProb);
   else if (timeUntilThreat == "imminent-current")
      totalProb.setValue(0.0);
}
if (PatrollingTroopsNear)
{
   if ( timeUntilThreat == "days")
      totalProb.setValue(0.2 * totalProb);
   else if (timeUntilThreat == "hours")
      totalProb.setValue(0.6 * totalProb);
   else if (timeUntilThreat == "imminent-current")
      totalProb.setValue(0.2 * totalProb);
}
if (StrikeForce)
{
   if ( timeUntilThreat == "days")
      totalProb.setValue(0.0);
   else if (timeUntilThreat == "hours")
      totalProb.setValue(0.2 * totalProb);
   else if (timeUntilThreat == "imminent-current")
      totalProb.setValue(0.8 * totalProb);
}
estimatedTimeThreatContactProbability.setValue(totalProb);
```

```
        }
        void Pause()
        {
        }
        void Resume()
        {
        }
        void Kill()
        {
            /* The following may be a self-generated call. User does not need to define it. If the
assumption
             * is correct, it would the call to the kill() method of this thread object.
             */
            dispose();
        }
    }

    activityGen ComputeEscapeDifficultyLevel()
    {

        void Initialize()
        {
        }
        void Main()
        {
            if (currentThreatIntensity == "high")
            {
                if (timeUntilThreat == "days")
    {
        escapeDifficultyProbability.M.setValue(.5);
        escapeDifficultyProbability.H.setValue(.5);
    }
    else if (timeUntilThreat == "hours")
    {
        escapeDifficultyProbability.H.setValue(1.0);
    }
    else if (timeUntilThreat == "imminent-current")
    {
        escapeDifficultyProbability.H.setValue(1.0);
    }
}
else if (currentThreatIntensity == "medium")
{
    if (timeUntilThreat == "days")
    {
        escapeDifficultyProbability.M.setValue(1.0);
    }
    else if (timeUntilThreat == "hours")
    {
        escapeDifficultyProbability.M.setValue(0.5);
        escapeDifficultyProbability.H.setValue(0.5);
    }
    else if (timeUntilThreat == "imminent-current")
    {
        escapeDifficultyProbability.H.setValue(1.0);
    }
    }
    else if (currentThreatIntensity == "low")
    {
        if (timeUntilThreat == "days")
        {
            escapeDifficultyProbability.L.setValue(1.0);
        }
        else if (timeUntilThreat == "hours")
        {
            escapeDifficultyProbability.L.setValue(0.5);
            escapeDifficultyProbability.M.setValue(0.5);
        }
        else if (timeUntilThreat == "imminent-current")
        {
            escapeDifficultyProbability.M.setValue(1.0);
        }
    }
}
void Pause()
{
}
void Resume()
{

}
void Kill()
{
        /* The following may be a self-generated call. User does not need to define it. If the
assumption
         * is correct, it would the call to the kill() method of this thread object.
         */
        dispose();
      }
   }

   activityGen LeaveTroopsBehind()
   {


      void Initialize()
      {
      }
      void Main()
      {
        if (escapeDifficulty == "high")
        {
           remainingLowQualityTroop.setValue(0);
           remainingMediumQualityTroop.setValue(remainingMediumQualityTroop / 2);
           remainingHighQualityTroop.setValue(remaingHighQualityTroop / 2);
        }
        else if (escapeDifficulty == "medium")
        {
           remainingLowQualityTroop.setValue((remainingLowQualityTroop * 1) / 4);
           remainingMediumQualityTroop.setValue(remainingMediumQualityTroop / 2);
           // remainingHighQualityTroop is not sacrificed
```

```
            }
            else if (escapeDifficulty == "low")
            {
                remainingLowQualityTroop.setValue(remainingLowQualityTroop / 2);
                remainingMediumQualityTroop.setValue((remainingMediumQualityTroop * 9) / 10);
                // remainingHighQualityTroop is not sacrificed
            }
        }
        void Pause()
        {

        }
        void Resume()
        {

        }
        void Kill()
        {
            /* The following may be a self-generated call. User does not need to define it. If the
assumption
             * is correct, it would the call to the kill() method of this thread object.
             */
            dispose();
        }
    }
}
```

Appendix C: Quasi-Executable Specification of a Human-Systems Simulation Experiment

Simulation scenario

The simulation is a man-hunting scenario in Afghanistan. The target of the man-hunting is a certain key figure, KF, that attempts to exit the country by crossing the border with Pakistan. KF is a drug lord that has been advised by his informants and associates in the government to exit the country and keep a low profile.

KF’s exit strategy is to attempt to reach Pakistan by land using the Southern routes. The southern border of Afghanistan with Pakistan is particularly porous, and KF has a great network of passes and secondary roads to choose from while selecting the route. KF plans to gather a small convoy of about three cars and some scouters. Although the convoy is heavily armed, KF seeks to avoid as much as possible any engagement with border guards, police, or coalition forces.

On the route to Pakistan, KF maintains several safe houses, often located in villages where the population shares his ethnicity and is willing to give him support and shelter. However, KF’s stay in a safe house cannot be too long: the longer he stays in Afghanistan, the more likely some coalition forces will locate and capture him.

The coaltion forces know the area to search for KF and has a network of sensors to locate him and informants to tip off about his location. In the operation, the coalition forces receive support from the Afghan police and border guards. However, they are concerned about the lack of proper equipment and about the corruption among the border guards and police. In fact, KF knows that, although he is unlikely to be successful in bribing or defeating some coalition forces, he is sure that the odds of success are substantially higher if he encounters police or border guards.

Simulation Parameters

Time and Execution

The simulation engine is responsible to run a clock. We define a certain amount of time ticks (say 2000) and set current time to zero and set end of simulation flag to zero.

For example, we have:

```
Integer Current_Time = 0;
Integer End_Time = 2000;
Boolean End_Simulation; End_Simulation.setValue (false);
```

Actors

The simulation contains a certain amount of actors, including the key figure and the "good guys". For example, we may specify 4 actors for the simulation:

```
Integer NumActors = 5;
```

There is only one type of key figure actor, but three types of good guys actors. E.g.,

```
String Actor_Type = “key figure”;       // Actor needs to have the key figure practices
String Actor_Type = “coalition”;        // type of good guy
String Actor_Type = “police”;                   // type of good guy
String Actor_Type = “border guard”;// type of good guy
```

Each actor has a name. E.g.,

```
String Actor_Name = “key figure”;
String Actor_Name = “coalition 1”;
String Actor_Name = “police 1”;
String Actor_Name = “police 2”;
String Actor_Name = “border guard 1”;
```

For each type of actor, except for “key figure”, we define three sizes, “large”, “med”, “small”. We assign an actor size for each actor. E.g.,:

```
String Actor_Name = “large”;            // for “coalition 1”
String Actor_Name = “med”;      // for “police 1”
String Actor_Name = “small”;            // for “police 1”
String Actor_Name = “small”;            // for “border guard 1”
```

Speed

For each actor, we assign two speed parameters, a regular speed and a fast speed, according to the following table:

Actor Type	***Regular_Speed***	***Fast_Speed***
Key Figure	36	60
Coalition	30	72
Police	24	42
Border guard	12	36

E.g.,

- for “key figure”

Real Regular_Speed = 36;

Real Fast_Speed = 60;

- for “coaltion 1”

Real Regular_Speed = 30;

Real Fast_Speed = 72;

Sensor Degeneration Rates

For each actor, the probability to detect an opponent depends on the distance D between the actor and the opponent. The longer the distance, the less likely that the detection will occur

- Range of 100% chance detection: D < ***range_100***;
- Range of 50% chance of detection: ***range_100*** <= D < ***range_50***
- Range of 25% chance of detection: ***range_50*** <= D < ***range_25***
- Range of 0% chance of detection: D > ***range_25***

We define the degeneration rates according to the values we set for range_25, range_50, and range_100. We specify the following table for the sensor degeneration rates:

Actor Type	***Range_100***	***Range_50***	***Range_25***
Key Figure	5	9	13
Coalition	5	9	13
Police	5	7	9
Border guard	5	6	7

E.g.,

- For “key figure”

Real Range_100 = 5;

Real Range_50 = 0;

Real Range_25 = 13;

Phone Call Probability

For each actor, there is an probability that the actor may receive a phone call with a precise location of an opponent. We specify the following table for the phone call probabilities:

Actor Type	***Phone_Call_Probability***
Key Figure	0.4
Coalition	0.3
Police	0.01
Border guard	0.01

Threat Tolerance for Key Figure

The key figure has several parameters that indicate their policy for threat tolerance. These variables are:

- Threat_Threshold: maximum amount of threat that the key figure tolerates
- Num_Cycles: number of threat information updates required to increse threshold for threat tolerance
- Threshold_Change_Rate: percentage increase each time the key figure decides to change threshold

We initiate these parameters with the following values:

Real Threat_Threshold = 0.25;

Integer Num_Cycles = 3;

Real Threshold_Change_Rate = 1.1;

Location Estimation Decay Rates for Good Guys

The location of the key figure is given by a estimate of its position plus an estimation range. At the time the actor receives the position of the key figure, the estimation range is zero (i.e. the position is an accurate estimation of its location). However, as times passes, the location become less accurate as the estimation range increases (i.e. the location is somewhere in the area centered in the last reported position within a certain radius. The decay rates represent the rate of that this radius increase per time tick. We specify the following table for the actors:

Actor Type	***Decay_Rates***
Coalition	2

Police	3
Border guard	5

E.g.,

- for “coalition 1”

Real Decay_Rate = 2;

Data Structures

Terrain_Map

Terrain_Map the “terrain map” of the scenario, i.e., the list of the possible ways that links any two points in the map.

There are two major elements in the map:

- waypoints (vertices)
- roads (edges)

Terrain_Map is a collection of roads, each road with a start waypoint, an end waypoint, and a length.

Format of a Terrain_Map element: (road_id, start_waypoint, end_waypoint, road_length)

With this format, a a certain road with <road_id> connects two points, <start_waypoint> and <end_waypoint>, and has has lenght <road_length>.

In the simulation, create your own terrain map. One example is given below.

```
Vector Terrain_Map =
((road1,     "origin",     "wp1",              3),
 (road2,     "wp1", "wp2",               15),
 (road3,     "wp2", "safe house 1",            2),
 (road4,     "wp1", "wp3",               10),
 (road5,     "wp2", "wp3",               6),
 (road6,     "wp1", "wp4",               12),
 (road7,     "wp4", "safe house 2",            4),
 (road8,     "wp3", "wp5",               14),
 (road9,     "wp2", "wp5",               20),
 (road10,    "wp5", "wp6",               8),
 (road11,    "wp6", "wp7",               8),
 (road12,    "wp6", "final destination",  7),
 (road13,    "wp5", "final destination",  11),
 (road14,         "wp7", "final destination",      13))
```

Aug_Terrain_Map

Aug_Terrain_Map is an augmented version of Terrain_Map.

In the augmented map, each road has two entries, the original entry with edge_direction = "forward" and a second entry, where we switch start_waypoint and end_waypoint and have edge_direction = "backward";

This table is defined simply to facilitate the use of Dijikstra algorithm to compute the shortest path in a directed graph (with vertices and edges). In this case, assigning a direction to a road makes it an edge (a directed link).

Format of Aug_Terrain_Map element: (road_id, start_waypoint, end_waypoint, road_cost, edge_direction)

Using Terrain_Map above, we have the respectve augmented map.

```
Vector Aug_Terrain_Map =
((road1,         "origin",       "wp1",               3,       "forward"),
 (road2,         "wp1", "wp2",                 15,      "forward"),
 (road3,     "wp2", "safe house 1",             2,      "forward"),
 (road4,     "wp1", "wp3",                10,     "forward"),
 (road5,     "wp2", "wp3",                6,      "forward"),
 (road6,     "wp1", "wp4",                12,     "forward"),
 (road7,     "wp4", "safe house 2",             4,      "forward"),
 (road8,     "wp3", "wp5",                14,     "forward"),
 (road9,     "wp2", "wp5",                20,     "forward"),
 (road10,    "wp5", "wp6",                8,      "forward"),
 (road11,    "wp6", "wp7",                8,      "forward"),
 (road12,    "wp6", "final destination",  7,      "forward"),
 (road13,    "wp5", "final destination",  11,     "forward"),
 (road14,    "wp7", "final destination",  13,     "forward"),
 (road1,     "wp1", "origin",                   3,      "backward"),
 (road2,     "wp2",  "wp1",                     15,     "backward"),
 (road3,     "safe house 1",               "wp2",      2,      "backward"),
 (road4,     "wp3",  "wp1",                     10,     "backward"),
 (road5,     "wp3",  "wp2",                     6,      "backward"),
 (road6,     "wp4",  "wp1",                     12,     "backward"),
 (road7,     "safe house 2",               "wp4",      4,      "backward"),
 (road8,     "wp5",  "wp3",                     14,     "backward"),
 (road9,     "wp5",  "wp2",                     20,     "backward"),
 (road10,    "wp6",  "wp5",                     8,      "backward"),
 (road11,     “wp7”, “wp6”,                        8,     “backward”),
 (road12,     “final destination”,   “wp6”,        7,     “backward”),
 (road13,     “final destination”,   “wp5”,        11,    “backward”),
 (road14,     “final destination”,   “wp7”,        13,    “backward”),);
```

Waypoints_List

Waypoints_List is a collection of all waypoints (vertices) of the Terrain_Map. Each waypoint has format of string. Observe that the origin, destination, and safe houses are all waypoints.

For example, the terrain map above contains the following waypoints.

Vector Waypoints_List = (“origin”, “final destination”, “safe house 1”, “safe house 2”, “wp1”, “wp2” “wp3”, “wp4”, “wp5”, “wp6”, “wp6”,);

Safe_Houses_List

Safe_Houses_List is a subset of Waypoints_List containing a list of all safe houses in the scenario, including the origin and final destination.

For example, the terrain map above contains the following safe houses.

Vector Safe_House_Lists = (“origin”, “final destination”, “safe house 1”, “safe house 2”);

Final_Destination

A string containing the final destination node for the key figure. In the example, the final destination is “final destination”

String Final_Destination = “final destination”;

Current_Position

Current_Position represents the current position of the actor in the map.

In our system of coordinates, a position is expressed in terms of the road where the point is located and the distance of that point to the start waypoint (i.e., the road position).

Format: (road_id, road_position)

E.g., (road2, 3) refers to a position in road2 located 3 km from “wp1”

In the simulation, we initiate the current positions for the actors as follows:

- **Key Figure**: Vector Current_Position = (road1, 0)

- **Good Guys**: Vector Current_Position = (random_Road (), random_Position ())

where the functions:

- ***Vector random_Road ()*** randomly select a road from road1 to road14
- ***Vector random_Position ()*** randomly select a position between 0 and the lenght of the selected road

Vector random_Position_In_Map () {}

returns a position (road_id, road_position) randomly selected from those available in the map. We simply randomly select a road (from road1 to road14) and randomly select a position (from 0 to the lenght of the road).

Next_Destination

Next_Destination contains the position of the next destination of a given actor. In the simulation, we initiate Next_Destination with the actor’s current position.

Vector Next_Destination = Current_Position;

Current_Route

Current_Route is a sequence of directed roads that link two positions in Terrain_Map, the Current_Position and the Next_Destination. Each element of Route contains a road and a direction.

Format of an element of Route: (road_id, road_direction).

We define two road directions:

- “forward”, that specifies the direction from start_waypoint to end_waypoint
- “backward”, that specifies the direction from end_waypoint to start_waypoint

For example, the following route links “origin” to “wp2”

Current_Route = ((road1, “forward”), (road4, “forward”), (road5, “backward”))

Notice that Current_Position must be a position in road1 (first route element) and Next_Destination must be a position in road5 (last route element).

In the simulation, initiate Current_Route as follows:

- **Key Figure**: Vector Current_Route = (road1, “forward”)
- **Good Guys**: Vector Current_Route = (random_Road (), “forward”)

Ground_Truth_Table

Ground_Truth_Table contains of the real position for each actor at each time tick of the simulation.

Format of each element: (threat_id, actor_name, actor_position). We iniate the table with the Current_Position of the each actor when they were created. For example,

```
Vector Ground_Truth_Table =
((not_threat,    "key figure",   (road0, 0)),
 (threat1,          "coalition 1",  (road5, 1)),    // randomly selected positions
 (threat2,          "police 1",     (road3, 4)),    // randomly selected positions
 (threat3,          "police 2",     (road7, 10),    // randomly selected positions
 (threat3,          "police 2",     (road7, 10));   // randomly selected positions
```

Threat_Map

Threat_Map is used by the key figure to track the position of each good guy at each time tick. In addition to the threat identification and position, each element of Threat_Map contains the location range for the position estimate and also the type and size of the threat (good guy).

Format of an element: (threat_id, threat_type, threat_size, threat_position, position_range).

We initiate the simulation with the initial position of the threats but assign a large number for position range (say, 50). For example, we may initiate the table as follows:

```
Vector Threat_Map =
 ((not_threat,   “key figure”,   “l”,                (road0, 0),       50),
 (threat1,                “coalition”,     “large”,          (road5, 1),       50),
 (threat2,                “police”,                 “med”,(road3, 4),       50),
 (threat3,                “police”,                 “small”,         (road7, 10),     50),
 (threat3,                “police”,                 “small”,         (road7, 10),     50);
```

Threat_Sensor_Data

Threat_Sensor_Data contains a list of messages informing the current position for several threats, as reported by its sensors.

Format of an element (threat_id, threat_position)

Threat_Risk_Table

Associate type and size of threat with its defeatability and bribability

Format of an element (t_type, t_size, defeat, brib). For example,

```
Vector Threat_Risk_Table =
((“Coalition”,       “Large”,    0.01,   0.01),
 (“Coalition”,       “Med”,      0.02,   0.01),
 (“Coalition”,       “Small”,    0.03,   0.01),
 (“Police”,          “Large”,    0.4,            0.5),
 (“Police”,          “Med”,      0.5,            0.5),
 (“Police”,          “Small”,    0.6,            0.5),
 (“Border guard”,    “Large”,    0.7,            0.8),
 (“Border guard”,    “Med”,      0.8,            0.8),
 (“Border guard”,    “Small”,    0.9,            0.8));
```

In this case, the key figure perceives 0.4 probability to defeat a large police and 0.5 probability to bribe it. On the other hand, the key figure perceives only a 0.02 to defeat a medium coalition and 0.01 to bribe it.

Algorithms

Route Selection

To select a route between two positions, we use Dijikstra algorithm for shortest path computation. The Dijikstra algorithm takes a collection of Vertices (nodes), a collection of Edges (directed links) connecting the vertices, and the cost of each edge as inputs. It returns the sequence of Edges from a given starting vertex to a given end vertex that minimizes the total costs of the edges traversed.

In other to match the inputs of the Dijikstra algorithm, we need to perform the following adaptations:

Origin and Destination

In Dijikstra algorithm, the start and end points are vertices. However, in our system of coordinates, a starting or ending point can be a position in the middle of an edge (road). In this case, we perform the following operations

- Convert the starting and ending points into two new vertices (start_v and end_v)
- Each of these new vertices represents a point that partition their respective road into two road sections: a section from the vertice to the start waypoint of the road and a section from the vertice to the end waypoint

<u>Vertices</u>

Given that the start and end points are re-defined as new vertices, we define the following set for the vertices

Vector Vertices = Waypoints_List + “start_v” + “end_v”;

Notice that we don’t need to add “start_v” (or “end_v”) if they are already waypoints.

Edges

Edges is based on Terrain_Map that lists a collection of roads connecting two points. However, an edge is a directed link, so we need to define a direction for the road. Thus, we define Aug_Terrain_Map for this purpose (see previous section).

In addition to the direction, we add some additional entries in Aug_Terrain_Map that correspond to the edges representing the new road sections that were created when we added the new vertices.

Costs

The costs are associated with each road, regardless of their direction. We have two types of costs, depending of what type of criteria we use for the Dijikstra algorithm.

For a simple shortest path computation, we use the length of the road (or the road section) as the cost.

For the “safest” path, we use a score system that scores the road (or road section) according to its length and according to the type and size of threat located in that road (or road section).

Road scoring

Roads are scored in terms of their cost. As mentioned before, the cost of a road (or road section) may be simply its length. More general, we may score the road as a function of its length and the threats located on it.

Thus, in general, we have

Score = Wd * D + Wt * T

D

D is the “relative length” of the road, that is, its fraction compared to the total distance that can be covered on the map. To obtain D, we divide the length of the road (or road section) by the sum of the lengths of all roads in Terrain_Map.

D = (length of the road or road section) / (sum of the lenghts of all roads in Terrain_Map)

T

T represents the probability that the actor will be captured in that road. It depends on the number of threats located on that road and the capture probability for each threat.

Thus, we have T = T1 * T2 * ... * Tn, where n is the number of threats on the road.

T_i (i = 1, 2, .., n) is a function of the threat defeatability ($Defeat_i$) and the threat bribability ($Bribe_i$). Defeatability is the probability that the actor is able to defeat the threat in a fight. Bribability is the probability that the threat will accept a bribe from the actor. To compute S_i, we use:

$$T_i = (1 - Bribe_i) * (1 - Defeat_i)$$

In other words, the actor will only be captured if the threat is not likely to accept a bribe and is not likely to be defeated in a fight with the actor.

Wd and Wt

Wd and Wt are the weights given to how important are the distance compared to threat level. In general, Wt is substantially larger than Wd (i.e. threat level is much more important than the distance). In our simulations, we set Wd and Wt to fixed values, such that Wd + Wt = 1 (in the case, Wt = 0.9 and Wd = 0.1), meaning that threat level is 9 times more important to score a road than the distance.

Suggestions for improvement

A more complex implementation of the scoring function would take advantage of the decay rate and estimation range of the location of the threat. Currently, we score a road according to whether there is a threat located on it. However, using the estimation range, we can say that the threat is somewhere within the threat position +/- estimation range. In addition, we can make the estimation range increase according to a certain decay rate as time goes by and the actor did not receive updates for the threat position.

In this case, as the estimation range increase, the threat will cover more roads, but its threat level on each road decreases.

Thus we suggest the following algorithm to compute the new threat level of the road (New_Ti).

For all threats in Threat_Map, compute the minimum distance to the road (minimum distance means the distance from the threat to the closest vertex of the road or road section). Call this distance Md_i.

Thus, we have that, for that road, $New\_T_i = T_i / Md_i$.

Threshold changes

When Route Planner of the key figure finds not acceptable routes, that is, there are no routes where the score is under Threat_Threshold, the key figure runs to the closest safe house. Once in the safe house, the General Manager initiates a process of looking at the past history of threat levels and change its threat tolerance depending upon this tendency.

For the current implementation, we use the following algorithm.

We select the last three update cycles while the key figure was in the safe house. An update cycle is a cycle where sensors capture new data, the threat analyzer maps the threat, and the route planner compute new safest path given the new threat map.

In each of these three cycles, we measure the best scores of the unaccepted routes. Let’s call them s1, s2, and s3, where s3 is the latest score.

We specify the following algorithm: if the average between s2 and s3 is higher than the average between s1 and s2, then the tendency is that the threat level is increasing, so the General Planner raises the threshold. However, if first average is lower than the second, the General Planner does nothing and wait until the threat level lowers to a level below its threshold.

Suggestions for improvement

A suggestion for improvement is to assess the threat level tendency by using more cycles (i.e. more than 3 cycles). In this case, instead of comparing averages, we do a linear regression and try to compute a line that best fit these points. If the slope of this line is positive, it means that the tendency is to increase the threat level, so the General Planner increases the threat tolerance. Otherwise, if the slope is negative, the General Planner waits for the threat level to decrease below the current threshold level.

Experiment Design

Independent Variables

1) Number of actors

   - Vary Num_Actors

2) Distribution of categories of good guys

   - Given a number of actors, specify how many coalitions, polices, or border guards and their sizes (large, medium or small)

3) Key Figure's Policy for Threat Tolerance

   - Vary Threat_Threshold
   - Vary Num_Cycles
   - Vary Threshold_Change_Rate

4) Key Figure's final destination

   - Change Terrain_Map (and, thus, Waypoints_List) and/or

- Select another waypoint as final destination

5) Number and location of Safe Houses

   - Change Terrain_Map and/or
   - Change Safe_Houses_Lists
   - Select other waypoints as safe houses

6) Sensor Degeneration Rates

   - Change Range_100, Range_50, and Range_25 for each actor

7) Location Estimation

   - Change Decay_Rates for each good guy

8) Probability of phone call with opponent position

   - Change Phone_Call_Probability for each actor

ACTOR A1: KEY FIGURE

Practice A1: Key Figure Route Planner

Information environment

```
// Simulation Environment
Integer Current_Time = 0;              // Track the current time tick of the simulation, init with 0
Integer End_Time = 2,000;              // Total number of time ticks, init with 2000
Boolean End_Simulation;                    // Route Planner uses to notify end of simulation
End_Simulation.setValue (false)        // Init with false

// Actor
String Actor_Id       = “key figure”;
String Actor_Type = “key figure”;

// Shared variables
String Final_Destination = “final destination”;
Vector Next_Destination = (road1, 0);
Vector Current_Position = Next_Destination;
Vector Current_Route = (road1, “forward”);

Real Speed = 0;

Vector Threat_Map;


Real Threat_Threshold = 0.25;

Real Best_Score_Cycle = Threat_Threshold;


// Flags

Boolean Threat_Map_Flag;          // flag if Threat_Map is updated

Threat_Map_Flag.setValue (false);

Boolean Safe_House_Mode;          // flag if Route_Planner returned route to safe house

Safe_House_Mode.setValue (false);

Boolean Cycle_Flag;               // flag if Route_Planner ran a cycle of threat data update

Cycle_Flag.setValue (false);

Boolean In_Safe_House;                    // flag if Current_Position is a safe house

In_Safe_House.setValue (false);

Boolean At_Destination;                   // flag if arrived at Next_Destination

At_Destination.setValue (true);

Boolean Target_Captured;          // flag if actor has been captured

Target_Captured.setValue (false);

Boolean Go_Flag;                          // flag Motion Manager to move

Go_Flag.setValue (false);
```

```
Boolean Target_Escaped;                 // flag if arrived at final destination

Target_Escaped.setValue (false);


// On-Off Flags for the practices, initiate all with true

Boolean Motion_Manager_On; Motion_Manager_On.setValue (true);

Boolean Gen_Planner_On; Gen_Planner_On.setValue (true);

Boolean Threat_Analyzer_On; Threat_Analyzer_On.setValue (true);

Boolean Threat_Sensor_On; Threat_Sensor_On.setValue (true);

Booleant Phone_Caller_On; Phone_Caller_On.setValue (true);


// Set types

set RoadT = {road1, road2, road3, road4, road5, road6, road7, road8, road9, road10, road11, road
12, road13, road14, r1a, r1b, r2a, r2b}

set ThreatT = {not_threat, threat1, threat2, threat3, threat4}


// Numeric Constants

Real Regular_Speed =36;       // see speed table

Real Fast_Speed = 60;                   // see speed table

Real Wd = 0.1;                                  // weight of the distance factor for scoring

Real Wt = 0.9;                          // weight of the threat level factor for scoring


// Vector Constants: see Appendix on how to initialize these tables
```

```
Vector Terrain_Map;

Vector Aug_Terrain_Map;

Vector Waypoints_List;

Vector Safe_Houses_Lists;

Vector Threat_Risk_Table;


// Constants representing indices: Terrain_Map, Aug_Terrain_Map, positions, Route

Integer road_id = 0; Integer start_waypoint = 1; Integer end_waypoint = 2; Integer road_length =
3; Integer road_cost = 3; Integer edge_direction = 4; Integer road_position = 1; Integer
road_direction = 1


// Constants representing indices: Threat_Map, Threat_Sensor_Map, Threat_Risk_Table,
Ground_Truth_Table

Integer threat_id = 0; Integer threat_position = 1; Integer threat_type = 2; Integer threat_size = 3;

Integer position_range = 4; Integer t_type = 0; Integer t_size = 1; Integer defeat = 2; Integer
bribe = 3;

Integer actor_name = 1; Integer actor_position = 2;


// Local

Vector Proposed_Route;
```

Monitors

```
monitor Monitor_Routing (Threat_Map_Flag, Best_Score_Cycle, In_Safe_House)
{
    If (Threat_Map_Flag == true) {
        propose_Route.Main ();                          // find best route to final destination

        If (Best_Score_Cycle <= Threat_Threshold) {     // if route acceptable, go to final destination
            go_To_Final_Destination.Main ();
            Safe_House_Mode.setValue (false);
            Go_Flag.setValue (true);
            }

        Else If (Best_Score_Cycle > Threat_Threshold)   // if not acceptable, go to safe house

            if (In_Safe_House == false) {
                go_To_Best_Safehouse.Main ();
                Safe_House_Mode.setValue (true);
                Go_Flag.setValue (true);
            }
```

```
                    Else
                              Go_Flag.setValue (false);

          Threat_Map_Flag = false;     // reset flag for new updates on threat locations
(Threat Analyzer)
          Cycle_Flag = true;                    // reset flag for new update cycle (General
Planner)
     }
}

monitor Monitor_Simulation_End (Current_Time, Target_Captured, Current_Position)
{
     Target_Escaped.setValue (is_Final_Destination (Current_Position));

     if ((Current_Time == End_Time) || (Target_Captured) || (Target_Escaped)) {
          End_Simulation.setValue (true);
          Motion_Manager_On.setValue (false);
          Gen_Planner_On.setValue (false);
          Threat_Analyzer_On.setValue (false);
          Threat_Sensor_On.setValue (false);
          Phone_Caller_On.setValue (false);
     }
```

```
}
```

Activity Generator

activityGen propose_Route

```
{
  void Initialize() {}

  void Main()
  {
        Proposed_Route = get_Safest_Route (Current_Position, Final_Destination);
        Best_Score_Cycle = route_Score (Route);
  }

  void Resume() {}
  void Pause() {}
  void Kill() {}
}
```

activityGen go_To_Final_Destination

```
{
  void Initialize(){}
  void Main()
  {
        Current_Route = Proposed_Route;                    // Commit to proposed route
        Next_Destination = get_Next_Destination_Coord (Current_Route);
        Speed = Regular_Speed;
  }

  void Resume() {}
  void Pause() {}
  void Kill() {}
}
```

activityGen go_To_Best_Safehouse

```
{
  void Initialize() {}

  void Main()
  {
        Current_Route = route_To_Safe_House ();
        Next_Destination = get_Next_Destination_Coord (Current_Route);
        Speed = Fast_Speed;
  }
  void Resume() {}

  void Pause() {}

  void Kill() {}

}
```

Practice A2: General Planner

Information environment

```
// Shared variables

Real Threat_Threshold;

Real Best_Score_Cycle;


// Flags

Boolean Cycle_Flag;

Boolean In_Safe_House;


// On-Off Flags

Boolean Gen_Planner_On;


// Local variables

Vector Best_Score_History;

Integer Safe_House_Cycles = 0;      // init = 0


// Local numeric Constants

Integer NumCycles;                  // Init = 3, see variables
```

```
Real Threshold_Change_Rate;          // Init = 1.1, see variables
```

Monitors

```
monitor Monitor_Cycle (Cycle_Flag, In_Safe_House, Safe_House_Cycles)
{
        if (Cycle_Flag) {

                track_Score_History.Main ();

                if (In_Safe_House) {
                        Safe_House_Cycles++;
                        adjust_Threshold.Main ();
                }
                else
                        Safe_House_Cycles = 0;

                Cycle_Flag = false;             // reset flag for next cycle
        }
}
```

```
monitor Monitor_On_Off  (Gen_Planner_On)

{

        if (Gen_Planner_On == false) {

                track_Score_History.Pause ();

                adjust_Threshold.Pause ();

        }

}
```

Activity Generator

```
activityGen track_Score_History

{

  void Initialize() {}



  void Main()

  {

        // Check if Best_Score_History reached NumCycles. If yes, remove earlist element (at 0);

        if (Best_Score_History.size() > NumCycles) Best_Score_History.removeElementAt (0);



        // insert most recent score at the end of the collection

        Best_Score_History.addElement (Best_Score);

}
```

```
  void Resume() {}

  void Pause() {}

  void Kill() {}

}
```

activityGen adjust_Threshold

```
{
  void Initialize() {}


  void Main()

  {

        If (Safe_House_Cycles > NumCycles) {

                Integer Last_Index = Best_Score_History.size() – 1;

                Real Last_Point = Best_Score_History.elementAt (Last_Index);

                Real Point _2 = Best_Score_History.elementAt (Last_Index – 1);

                Real Point _3 = Best_Score_History.elementAt (Last_Index – 2);


                Real Avg1 = (Last_Point + Point_2) / 2;

                Real Avg2 = (Point_2 + Point_3) / 2;


                if (Avg2 >= Avg1) Threat_Threshold = Threshold_Increase_Range *
Threat_Threshold;
```

```
            }
    }


       void Resume() {}
       void Pause() {}
       void Kill() {}
    }
```

Practice A3: Key Figure Motion Manager

Information environment

```
// Actor
String Actor_Id                          // “key figure”


// Shared variables
Vector Next_Destination;
Vector Current_Position;
Vector Current_Route;
Vector Ground_Truth_Table; // init from Appendix
Real Speed;


// Flags
Boolean In_Safe_House;
Boolean At_Destination;
Boolean Go_Flag;


// On-Off Flags
Boolean Motion_Manager_On;
```

// Sets

set RoadT = {road1, road2, road3, road4, road5, road6, road7, road8, road9, road10, road11, road 12, road13, road14, r1a, r1b, r2a, r2b}

set ThreatT = {not_threat, threat1, threat2, threat3, threat4}

// Numeric Constants

Real Regular_Speed;

Real Fast_Speed;

// Vector Constants: see Appendix

Vector Terrain_Map;

Vector Aug_Terrain_Map;

Vector Waypoints_List;

Vector Safe_Houses_Lists;

// Constants representing indices: Terrain_Map, Aug_Terrain_Map, positions, Route

Integer road_id = 0; Integer start_waypoint = 1; Integer end_waypoint = 2; Integer road_length = 3; Integer road_cost = 3; Integer edge_direction = 4; Integer road_position = 1; Integer road_direction = 1

// Constants representing indices: Threat_Map, Threat_Sensor_Map, Threat_Risk_Table, Ground_Truth_Table

```
Integer threat_id = 0; Integer threat_position = 1; Integer threat_type = 2; Integer threat_size = 3;

Integer position_range = 4; Integer t_type = 0; Integer t_size = 1; Integer defeat = 2; Integer
bribe = 3;

Integer actor_name = 1; Integer actor_position = 2;


// Local

Boolean No_Motion; No_Motion.setValue (false);   // flag if Motion Manger is paused
```

Monitors

```
monitor Monitor_Manager_On_Off (Motion_Manager_On)
{
        if (Motion_Manager_On)
                move_Key_Figure.Main ();
}


monitor Monitor_Motion (Go_Flag, No_Motion)
{
        if (Go_Flag == false)
                move_Key_Figure.Pause ();
        else if (No_Motion) {
                move_Key_Figure.Resume ();
        }
}
```

Activity Generator

activityGen move_Key_Figure

```
{
  void Initialize() {}


  void Main()
  {
        Real Position_Inc = Speed / 60;              // calculate position increment in one time tick given speed


        Real Dest_Distance = route_Distance (Current_Position, Next_Destination, Current_Route);


        // if destination is within Position_Inc, mark that reached destination and check if in safe house
        If (Dest_Distance <= Position_Inc) {
                Current_Position = Next_Destination;
                At_Destination.setValue (true);
                if (is_Safe_House_Position (Current_Position)) In_Safe_House.setValue (true);
        }
        Else Current_Position = find_Next_Position (Dest_Distance, Current_Position,
Current_Route);

        update_Truth_Table_Position (Actor_id, Current_Position);
}

  void Resume()
  {
        No_Motion.setValue (false);
  }

  void Pause()
  {
        No_Motion.setValue (true);
  }

  void Kill() {}
}
```

Practice A4: Threat Analyzer

Information environment

```
// Shared variables

Vector Threat_Sensor_Data;  // initially empty

Vector Threat_Map;                    // see Appendix for initialization


// Flags

Boolean Threat_Map_Flag;


// On-Off Flags

Boolean Threat_Analyzer_On;


// Sets

set RoadT = {road1, road2, road3, road4, road5, road6, road7, road8, road9, road10, road11, road
12, road13, road14, r1a, r1b, r2a, r2b}

set ThreatT = {not_threat, threat1, threat2, threat3, threat4}


// Constants representing indices: Terrain_Map, Aug_Terrain_Map, positions, Route
```

Integer road_id = 0; Integer start_waypoint = 1; Integer end_waypoint = 2; Integer road_length = 3; Integer road_cost = 3; Integer edge_direction = 4; Integer road_position = 1; Integer road_direction = 1

// Constants representing indices: Threat_Map, Threat_Sensor_Map, Threat_Risk_Table, Ground_Truth_Table

Integer threat_id = 0; Integer threat_position = 1; Integer threat_type = 2; Integer threat_size = 3;

Integer position_range = 4; Integer t_type = 0; Integer t_size = 1; Integer defeat = 2; Integer bribe = 3;

Integer actor_name = 1; Integer actor_position = 2;

Monitors

```
monitor Monitor_Threat_Sensor_Data (Threat_Sensor_Data)

{

        If (Threat_Sensor_Data.size () > 0)

                process_Messages.Main ();

}


monitor Monitor_Analyzer_On_Off (Threat_Analyzer_On)

{

        If (Threat_Analyzer_On == false)

                process_Messages.Pause ();
}
```

Activity Generator

activityGen process_Messages

```
{
  void Initialize() {}


  void Main()
  {
        Vector Message;


        // Read messages and update Threat_Map
        Integer i = 0;
        while (i < Threat_Sensor_Data.size()) {
                Message = Threat_Sensor_Data.elementAt (i);
                process_A_Message (Message, Threat_Map);
                i++;
        }


        // clear message table
        i = Threat_Sensor_Data.size () – 1;
```

```
            while (i >= 0) {
                    Threat_Sensor_Data.removeElementAt (i);
                    i--;
            }
            Threat_Map_Flag.setValue(true);
    }

      void Resume() {}
      void Pause() {}
      void Kill() {}
    }
```

Practice A5: Threat Sensor

Information environment

// Shared variables

Vector Current_Position;

Vector Threat_Sensor_Data;

Vector Ground_Truth_Table;

// On-Off Flags

Sensor_On;

// Sets

set RoadT = {road1, road2, road3, road4, road5, road6, road7, road8, road9, road10, road11, road 12, road13, road14, r1a, r1b, r2a, r2b}

set ThreatT = {not_threat, threat1, threat2, threat3, threat4}

// Vector Constants: see Appendix

Vector Terrain_Map;

Vector Aug_Terrain_Map;

Vector Waypoints_List;

Vector Safe_Houses_Lists;

Vector Threat_Risk_Table;

// Constants representing indices: Terrain_Map, Aug_Terrain_Map, positions, Route

Integer road_id = 0; Integer start_waypoint = 1; Integer end_waypoint = 2; Integer road_length = 3; Integer road_cost = 3; Integer edge_direction = 4; Integer road_position = 1; Integer road_direction = 1

// Constants representing indices: Threat_Map, Threat_Sensor_Map, Threat_Risk_Table, Ground_Truth_Table

Integer threat_id = 0; Integer threat_position = 1; Integer threat_type = 2; Integer threat_size = 3;

Integer position_range = 4; Integer t_type = 0; Integer t_size = 1; Integer defeat = 2; Integer bribe = 3;

Integer actor_name = 1; Integer actor_position = 2;

Monitors

```
monitor Monitor_Sensor (Sensor_On)

{

        if (Sensor_On)

                scan_Threats.Main ();

        else

                scan_Threats.Pause ();

}
```

Activity Generator

activityGen scan_Threats

```
{
  void Initialize() {}


  void Main()
  {
        Vector GT_Elem;
        Vector Position;
        Real D;
        Boolean Detected;

        Integer i = 1;          // Start with second element of Ground_Truth_Table -> the first is
key figure
        While (i < Ground_Truth_Table.size()) {
                GT_Elem = Ground_Truth_Table.elementAt(i);
                Position = GT_Elem.elementAt (actor_position);
                D = distance (Current_Position, Position);

                if (D < range_100) Detected = "true";
                else if ((D >= range_100) and (D < range_50)) Detected = random_Binary (0.5);
                else if ((D >= range_50) and (D < range_25)) Detected = ***random_Binary*** (0.25);
                else Detected = “false”;

                if (Detected == “true”) ***send_Message*** (GT_Elem.elementAt(threat_id), Position);
                i++;
            }
    }

     void Resume() {}
     void Pause() {}
     void Kill() {}
    }
```

Practice A6: Threat Phone Caller

Information environment

```
// Shared variables

Vector Threat_Sensor_Data;

Vector Ground_Truth_Table;

Vector Current_Position;


// On-Off Flags

Boolean Phone_Caller_On;


// Local Numerical constants

Real Phone_Call_Probability =  0.4;  // see table
```

Monitors

```
monitor Monitor_Phone_Caller (Bool Phone_Caller_On)

{

        if (Phone_Caller_On)

                scan_Threats.Main ();
        else
                ***scan_Threats.Pause ()***;
}
```

Activity Generator

activityGen scan_Threats

```
{
  void Initialize() {}


  void Main()
  {
        Vector GT_Elem;
        Vector Position;
        Real D;
        Boolean Detected;


        Integer i = 1;          // Start with second element of Ground_Truth_Table -> the first is
key figure
        while (i < Ground_Truth_Table.size()) {
                GT_Elem = Ground_Truth_Table.elementAt(i);
                Position = GT_Elem.elementAt (actor_position);
                Detected = random_Binary (Phone_Call_Probability);
                if (Detected == "true") send_Message (GT_Elem.elementAt (threat_id),
position);
                i++;
        }
}

  void Resume() {}
  void Pause() {}
  void Kill() {}
}
```

ACTOR B: GOOD GUYS

Practice B1: Good Guys Route Planner

Information environment

// Simulation Environment

Integer Current_Time = 0; // Track the current time tick of the simulation, init with 0

Integer End_Time = 2,000; // Total number of time ticks, init with 2000

Boolean End_Simulation; // Route Planner uses to notify end of simulation

End_Simulation.setValue (false) // Init with false

// Actor

String Actor_id = “coalition 1”; // or “police 1”, “police 2”, “border guard 1”, etc. See simulation settings

String Actor_Type = “coalition”; // or “police”, “border guard”

String Actor_Size = “large”; // or “med”, “small”

// Shared variables

Vector Next_Destination = (road5, 1); // initialize with a random point. See appendix

```
Vector Current_Position = Next_Destination;

Vector Current_Route = (road5, “forward”);        // Make sure the road is the same from Next_Destination

Real Speed = 0;


Vector Ground_Truth_Table;

Vector Target_Position_Estimate = Next_Destination;      // current estimate for target location, start with current position

Real Target_Position_Range = 50;                  // range surrounding target location, indicate location accuracy


// Flags

Boolean At_Destination;                 // flag if arrived at Next_Destination

At_Destination.setValue (true);

Boolean Target_Captured;          // flag if actor has been captured

Target_Captured.setValue (false);

Boolean Go_Flag;                        // flag Motion Manager to move

Go_Flag.setValue (false);

Boolean Target_Escaped;           // flag if arrived at final destination

Target_Escaped.setValue (false);

Boolean New_Update;                     // flag if there is some update from either sensor or phone call

New_Update.setValue (false);
```

```
// On-Off Flags for the practices, initiate all with true

Boolean Motion_Manager_On; Motion_Manager_On.setValue (true);

Boolean Sensor_Reader_On; Sensor_Reader_On.setValue (true);

Boolean Phone_Answerer_On; Phone_Answerer_On.setValue (true);

Boolean Pos_Conflict_Res_On; Pos_Conflict_Res_On.setValue (true);

Boolean Target_Sensor_On;

Booleant Phone_Caller_On; Phone_Caller_On.setValue (true);


// Set types

set RoadT = {road1, road2, road3, road4, road5, road6, road7, road8, road9, road10, road11, road
12, road13, road14, r1a, r1b, r2a, r2b}

set ThreatT = {not_threat, threat1, threat2, threat3, threat4}


// Numeric Constants

Real Regular_Speed = 30;    // see speed table

Real Fast_Speed = 72;             // see speed table

Integer Decay_Rate = 2;           // increase in the location range per time tick when actor
receives no location updates

Real Capture_Range = 1;     // good guy captures target if their distance < Capture_Range


// Vector Constants: see Appendix on how to initialize these tables
```

```
Vector Terrain_Map;

Vector Aug_Terrain_Map;

Vector Waypoints_List;

Vector Safe_Houses_Lists;

Vector Threat_Risk_Table;


// Constants representing indices: Terrain_Map, Aug_Terrain_Map, positions, Route

Integer road_id = 0; Integer start_waypoint = 1; Integer end_waypoint = 2; Integer road_length =
3; Integer road_cost = 3; Integer edge_direction = 4; Integer road_position = 1; Integer
road_direction = 1


// Constants representing indices: Threat_Map, Threat_Sensor_Map, Threat_Risk_Table,
Ground_Truth_Table

Integer threat_id = 0; Integer threat_position = 1; Integer threat_type = 2; Integer threat_size = 3;

Integer position_range = 4; Integer t_type = 0; Integer t_size = 1; Integer defeat = 2; Integer
bribe = 3;

Integer actor_name = 1; Integer actor_position = 2;


// Local

Boolean Target_Located;
```

Monitors

```
monitor Monitor_Routing (New_Update, At_Destination)
{
        if (New_Update)
                go_To_Target.Main ();


        If (At_Destination) {
                go_To_Random_Destination.Main ();
}


monitor Monitor_Travel_Mode (Target_Position_Range)


        Target_Located = is_Position_Accurate (Target_Position_Range);


        if (Target_Located)
                pursue.Main ();
        else
                patrol.Main ();
}


monitor Monitor_Simulation_End (Current_Time, Target_Captured, Current_Position)
```

```
{
        Target_Escaped.setValue (is_Target_Captured (Current_Position));


        if ((Current_Time == End_Time) || (Target_Captured) || (Target_Escaped)) {
                End_Simulation.setValue (true);
                Motion_Manager_On.setValue (false);
                Sensor_Reader_On.setValue (false);
                Phone_Answerer_On.setValue (false);
                Pos_Conflict_Res_On. setValue (false);
                Target_Sensor_On.setValue (false);
                Phone_Caller_On.setValue (false);
        }
}
```

Activity Generator

activityGen go_To_Target

```
{
  void Initialize() {}
  void Main()
  {
        Next_Destination = Target_Position_Estimate;
```

```
          Current_Route = get_Shortest_Route (Current_Position, Next_Destination);
  }


  void Resume() {}
  void Pause() {}
  void Kill() {}
}
```
activityGen go_To_Random_Destination
```
{
  void Initialize() {}


  void Main()
  {
          RoadT Cur_Road = Current_Position.elementAt (road_id);
          Real Cur_Road_Pos = Current_Position.elementAt (road_position);
          Real Road_Len = get_Road_Length (Cur_Road);


          Vector Road_List = get_Near_Roads (Cur_Road);


          Integer N = Road_List.size();
          RoadT Next_Road = Road_List.elementAt (rand_Int (0, N – 1));
          Real Next_Road_Pos = rand_Int (0, get_Length_Road (Next_Road));


          Next_Destination.setElementAt (road_id, Next_Road);

          Next_Destination.setElementAt (road_position, Next_Road_Position);


          Current_Route = get_Shortest_Route (Current_Position, Next_Destination);

          At_Destination.setValue (false);

   }


   void Resume() {}

   void Pause() {}

   void Kill() {}

}
```

activityGen pursuit

```
{

   void Initialize() {}

   void Main()

   {

          Speed = Fast_Speed;

   }


   void Resume() {}
```

```
  void Pause() {}

  void Kill() {}

}
```

activityGen patrol

```
{

  void Initialize() {}



  void Main()

  {

          Speed = Regular_Speed;

  }



  void Resume() {}

  void Pause() {}

  void Kill() {}

}
```

Practice B2: Good Guys Motion Manager

Information environment

```
// Actor
String Actor_id;                    // "coalition 1"


// Shared variables
Vector Next_Destination;
Vector Current_Position;
Vector Current_Route;
Real Speed;
Vector Ground_Truth_Table;


// Flags
Boolean Target_Captured;            // init with false
Boolean At_Destination;


// On-Off Flags
Boolean Motion_Manager_On;
```

```
// Numeric Constants

Real Regular_Speed;

Real Fast_Speed;

Real Capture_Range;


// Sets

set RoadT = {road1, road2, road3, road4, road5, road6, road7, road8, road9, road10, road11, road
12, road13, road14, r1a, r1b, r2a, r2b}

set ThreatT = {not_threat, threat1, threat2, threat3, threat4}


// Vector Constants: see Appendix

Vector Terrain_Map;

Vector Aug_Terrain_Map;

Vector Waypoints_List;

Vector Safe_Houses_Lists;

Vector Threat_Risk_Table;


// Constants representing indices: Terrain_Map, Aug_Terrain_Map, positions, Route

Integer road_id = 0; Integer start_waypoint = 1; Integer end_waypoint = 2; Integer road_length =
3; Integer road_cost = 3; Integer edge_direction = 4; Integer road_position = 1; Integer
road_direction = 1
```

// Constants representing indices: Threat_Map, Threat_Sensor_Map, Threat_Risk_Table, Ground_Truth_Table

Integer threat_id = 0; Integer threat_position = 1; Integer threat_type = 2; Integer threat_size = 3;

Integer position_range = 4; Integer t_type = 0; Integer t_size = 1; Integer defeat = 2; Integer bribe = 3;

Integer actor_name = 1; Integer actor_position = 2;

Monitors

```
Monitor_Motion_Manager_Status

{

        if (Motion_Manager_On)

                move_GG.Main ();

        else

                move_GG.Pause ();

}
```

Activity Generator

activityGen move_GG

```
{

  void Initialize() {}
  void Main()
  {
        Real Position_Inc = Speed / 60;              // calculate position increment in one time
tick given speed

        Real Dest_Distance = route_Distance (Current_Position, Next_Destination,
Current_Route);

        if (Dest_Distance <= Position_Inc) {
                Current_Position = Next_Destination;
                At_Destination.setValue (true);
        }
        Else Current_Position = find_Next_Position (Dest_Distance, Current_Position,
Current_Route);
        update_Truth_Table_Position (Actor_id, Current_Position);
  }

  void Resume() {}
  void Pause() {}
  void Kill() {}
}
```

Practice B3: Target Sensor Reader

Information environment

```
// Shared Variables

Vector Current_Position;

Vector Target_Position_Estimate;

Real Decay_Rate;


Vector Sensor_Report;                    // position of the target from sensor


// Flag

Boolean New_Update;

Boolean New_Sensor_Update;               // indicate an update of target position from sensor

New_Sensor_Update.setValue (false);


// On-Off Flag

Boolean Sensor_Reader_On;

Boolean Resolver_Go;

Boolean Go_Sensor;
```

Monitors

```
Monitor_Sensor_Reader_On (Sensor_Reader _On, New_Sensor_Update,
Target_Position_Range)
{
        decay (Target_Position_Range);
        New_Update = “false”;

        If (Sensor_Reader _On) {
                if (New_Sensor_Update) {
                        while (Resolver_Go_Sensor == false) {};     // wait until go from resolver
                        if (Go_Sensor)
                                check_Sensor.Main ();
                }
        Else
                check_Sensor.Pause ();
}
```

Activity Generator

```
activityGen check_Sensor
{
  void Initialize() {}
  void Main()
  {
        if (Sensor_Report.size() > 0) {
                Target_Position_Estimate = removeElementAt (Sensor_Report);
                Target_Position_Range = 0;
                New_Update.setValue(true);
                New_Sensor_Update.setValue(false);
        }
 }

  void Resume() {}
  void Pause() {}
  void Kill() {}
}
```

Practice B4: Target Phone Answerer

Information environment

```
// Shared Variables
Vector Current_Position;
Vector Target_Position_Estimate;
Real Decay_Rate;

Vector Phone_Report;            // position of the target from phone caller

// Flag
Boolean New_Update;
Boolean New_Phone_Call;             // indicate an update of target position from phone caller
New_Phone_Call.setValue (false);

// On-Off Flag
Boolean Phone_Answerer_On;
Boolean Resolver_Go;
Boolean Go_Phone;
```

Monitors

```
Monitor_Phone_Answerer_On          (Phone_Answerer_On,          New_Phone_Call,
Target_Position_Range)

{

    New_Update = "false";


    If (Phone_Answerer_On) {

        if (New_Phone_Call) {


            while (Resolver_Go == false) {};    // wait until go from resolver

            if (Go_Phone)

                check_Phone.Main ();

            }

        }

    Else

        check_Phone.Pause ();

}
```

Activity Generator

activityGen check_Phone

```
{
  void Initialize() {}


  void Main()

  {

        if (Phone_Report.size () > 0) {

                Target_Position_Estimate = removeElementAt (Phone_Report);

                Target_Position_Range = 0;

                New_Update.setValue (true);

                New_Phone_Call.setValue(false);

        }

 }


  void Resume() {}

  void Pause() {}

  void Kill() {}

}
```

Practice B5: Position Conflict Resolver

Information environment

```
// Shared Variables

Vector Current_Position;

Vector Target_Position_Estimate;

Vector Sensor_Report;

Vector Phone_Report;


// Flag

Boolean New_Update;

Boolean Resolver_Go;              // flag the sensor and phone answerer to wait for go

Resolver_Go.setValue (false);

Boolean Go_Sensor;           // flag the sensor reader to go

Go_Sensor.setValue (false);

Boolean Go_Phone;            // flag the telephone answerer to go

Go_Phone.setValue (false);


// On-Off Flag

Boolean Pos_Conflict_Res_On;
```

Monitors

```
Monitor_Conflict (Pos_Conflict_Res_On, New_Update,  Sensor_Report, Phone_Report)

{

    If (Pos_Conflict_Res _On) {

        if ((New_Phone_Call) && (New_Sensor_Update))



            if (Sensor_Report != Phone_Report)) {

                resolve_Conflicting_Pos.Main ();

        else {

            Go_Sensor.setValue (true);

            Go_ Phone.setValue (true);

        }

    }

    Else

        resolve_Conflicting_Pos.Pause ();



    Resolver_Go.setValue (true);

}
```

Activity Generator

activityGen resolve_Conflicting_Pos

```
{
   void Initialize() {}


   void Main()
   {
            // resolution policy can be complex
            // simplest solution: always choose sensor report
            Go_Sensor.setValue (true);
            Go_ Phone.setValue (false);
   }


   void Resume() {}
   void Pause() {}
   void Kill() {}
}
```

Practice B6: Target Sensor

Information environment

```
// Shared variables

Vector Sensor_Report;

Vector Ground_Truth_Table;

Vector Current_Position;


// Flags

Boolean New_Sensor_Update;


// On-Off Flags

Boolean Target_Sensor_On;


// Sets

set RoadT = {road1, road2, road3, road4, road5, road6, road7, road8, road9, road10, road11, road
12, road13, road14, r1a, r1b, r2a, r2b}

set ThreatT = {not_threat, threat1, threat2, threat3, threat4}


// Vector Constants: see Appendix
```

```
Vector Terrain_Map;

Vector Aug_Terrain_Map;

Vector Waypoints_List;

Vector Safe_Houses_Lists;

Vector Threat_Risk_Table;


// Constants representing indices: Terrain_Map, Aug_Terrain_Map, positions, Route

Integer road_id = 0; Integer start_waypoint = 1; Integer end_waypoint = 2; Integer road_length =
3; Integer road_cost = 3; Integer edge_direction = 4; Integer road_position = 1; Integer
road_direction = 1


// Constants representing indices: Threat_Map, Threat_Sensor_Map, Threat_Risk_Table,
Ground_Truth_Table

Integer threat_id = 0; Integer threat_position = 1; Integer threat_type = 2; Integer threat_size = 3;

Integer position_range = 4; Integer t_type = 0; Integer t_size = 1; Integer defeat = 2; Integer
bribe = 3;

Integer actor_name = 1; Integer actor_position = 2;
```

Monitors

```
Monitor_Target_Sensor_On (Target_Sensor_On)

{

        if (Target_Sensor_On)
```

```
            report_Location.Main ();
        else
            report_Location.Pause ();
}
```

Activity Generator

activityGen report_Location

```
{
  void Initialize() {}


  void Main()
  {
        Vector Targ_Position = get_Truth_Table_Position ("key figure");
        Boolean Detected;
        Real D;


        if (is_Safe_House_Position (Targ_Position))
            Detected = "false"
        else {
            D = distance (Current_Position, Key_Figure_Position);
```

```
                if (D < range_100) Detected = “true”;
                else if ((D >= range_100) and (D < range_50)) Detected = random_Binary (0.5);
                else if ((D >= range_50) and (D < range_25)) Detected = random_Binary (0.25);
                else Detected = “false”;
        }

        if (Detected) {
                Sensor_Report = Targ_Position;
                New_Sensor_Update.setValue(true);
        }
 }

  void Resume() {}
  void Pause() {}
  void Kill() {}
}
```

Practice B7: Target Phone Caller

Information environment

```
// Shared Variables

Vector Current_Position;

Vector Phone_Report;

Vector Ground_Truth_Table;


// Flags

Boolean New_Phone_Call;


// On-Off Flags

Boolean Phone_Caller_On;


// Numerical Constant

Real Phone_Call_Probability;
```

Monitors

```
Monitor_Phone_Caller_On (Phone_Caller_On)

{
        if (Phone_Caller_On)
                report_Location.Main ();
        else
                report_Location.Pause ();
}
```

Activity Generator

activityGen report_Location

```
{
  void Initialize() {}


  void Main()
  {
        Vector Targ_Position = get_Truth_Table_Position ("key figure");
        Boolean Detected


        Detected = random_Binary (Phone_Call_Prob);
        if (Detected == "true") {
                Phone_Report = Targ_Position;
                New_Phone_Call.setValue (true);
        }
}
```

/ PROCEDURES KEY FIGURE ROUTE PLANNER ****/***

```
// is_Final_Destination: check if Position is Final Destination

Boolean is_Final_Destination (Vector Position)
{
        RoadT Road = Position.elementAt (road_id);
        Real Road_Pos = Position.elementAt (road_position);
        Vector Map_Elem;

        Integer i = 0;
        while (i < Terrain_Map.size ()) {
                Map_Elem = Terrain_Map.elementAt (i);
                if (Map_Elem.elementAt (end_waypoint) == “final destination”) {
                        if ((Map_Elem.elementAt (road_id) == Road) && (Map_Elem.elementAt
(road_length) == Road_Pos))
                                return true;
                }
                i++;
        }
        return false;
}
```

/*** get_Safest_Route: return a route from Position to a waypoint that minimizes threat and distance ***/

```
Vector get_Safest_Route (RoadT Position, String Wp)
{
        RoadT Road = Position.elementAt(road_id);

        Real Road_Pos = Position.elementAt (road_pos);


        // get the vertices (waypoints) of the road

        String S = startWp (Road));

        String E = endWp (Road);


        // Buid Vertices and insert "start_v"

        Vector Vertices = Waypoints_List.copy ();

        Vertices.addElement ("start_v");


        // Build Edges

        Vector Edges = Aug_Terrain_Map.copy ();


        // Update the cost values for each edge

        Vector Edge_Elem;

        Real Score;
```

```
Integer i = 0;

while (i < Edges.size()) {

        Edge_Elem = Edges.elementAt(i);

        Score = score_Road (Edge_Elem.elementAt(road_id));

        Edge_Elem.setElementAt (road_cost, Score);

        Edges.setElementAt (i, Edge_Elem);

        i++;

}


// Insert new edges with new scores

Score = score_Road_Section (Road, Road_Pos, 1);

Edges.addElement (new_Edge (r1a, S, “start_v”, Score, “forward”));

Edges.addElement (new_Edge (r1a, ”start_v”, S, Score, “backward”));

Score = score_Road_Section (Road, Road_Pos, 2);

Edges.addElement (new_Edge (r1b, ”start_v”, E, Score, “forward”));

Edges.addElement (new_Edge (r1b, E, “start_v”, Score, “backward”);


Vector Route;

Route = shortest_Route_Dijisktra (“start_v”, Wp, Vertices, Edges));


// Replace r1a and r2a for Road

Vector Route_Elem;
```

```
         i = 0;

        while (i < Route.size()) {

                Route_Elem = Route.elementAt(i);

                if ((Route_Elem.elementAt(road_id) == r1a) || (Route_Elem.elementAt(road_id)
== r1b)) {

                        Route_Elem.setElementAt (road_id, Road);

                        Route.setElementAt (i, Route_Element);

                }

                i++;

        }

        return Route;

}
```

/*** route_Score: return the score (function of threat levels and distance) of a route starting from Position ***/

```
Real route_Score (Vector Position, Vector Route)

{

        Real Score = 0;

        Vector Edge = Route.elementAt(0);

        RoadT Road = Edge.elementAt (road_id);
        if (Edge.elementAt(edge_direction) == “forward”)
                Score = (***get_Road_Length*** (Road) – Position.elementAt (road_position));
        Else Score = Position.elementAt (road_position);


        i = 1;
        while (i < Route.size ()) {
                Edge = Route.elementAt(i);
                        Score = Score + ***Score_Road*** (Edge.elementAt(road_id));
                i++;
        }
        return Score;
}
```

/*** get_Next_Destination_Coord_KF: return the position (road_id, road_position) of the last point of the route ***/

```
Vector get_Next_Destination_Coord (Vector Route)
{
        Vector Route_Elem = Route.elementAt (Route.size() – 1);            //            Final
destinaton at last element
        RoadT Road = Route_Elem.elementAt(road_id);
        Real Road_Pos;
        Vector Next_Pos;
        // Next_Destination, for key figure, is always waypoint, so road_position is either 0 or the
road length

        if (Route_Elem.elementAt(road_direction) == “forward”) Road_Pos = get_Road_Length
(road);

        Else Road_Pos = 0;


        Next_Pos.insertElementAt(road_id, Road);

        Next_Pos.insertElementAt(road_position, Road_Pos);

        return Next_Pos;

}
```

/*** route_To_Safe_House: return the safest route of all routes to all safe houses (including origin and destination) ***/

```
Vector route_To_Safe_House ()
{

        Vector Best_Route, Route;

        Real Min_Score = 10000;

        Real S;

        String SH;


        // Find safest route for each safe house, and select the shortest of all
```

```
    Integer i = 0;

    while (i < Safe_Houses_List.size()) {

        SH = Safe_House_List.elementAt(i);

        Route = get_Safest_Route (Current_Position, SH);

        S = route_Score (Current_Position, Route);

        if (S < Min_Score) {

            Min_Score = S;

            Best_Route = Route;

        }

        i++;

    }

    return Best_Route;

}


/*** endWP: look up Terrain_Map and return the end_waypoint of Road ***/


String endWp (RoadT Road)
{

    Vector Terrain_Elem;


    Integer i = 0;

    while (i < Terrain_Map) {
```

```
                Terrain_Elem = Terrain_Map.elementAt (i);

                if (Terrain_Elem.elementAt (road_id) == Road) return Terrain_Elem.elementAt
(end_waypoint);

                i++;

        }

}


/*** startWP: look up Terrain_Map and return the start_waypoint of Road ***/


String startWp (RoadT Road)

{

        Vector Terrain_Elem;



        i = 0;

        while (i < Terrain_Map) {

                Terrain_Elem = Terrain_Map.elementAt (i);

                if (Terrain_Elem.elementAt (road_id) == Road) return Terrain_Elem.elementAt
(start_waypoint);

                i++;

        }

}


/*** new_Edge: format an edge to be entered in Edges ***/
```

```
Vector new_Edge (RoadT Road, String Start_v, String End_v, Real Score, String Direction))
{
        Vector Edge;
        Edge.insertElementAt(road_id, Road);
        Edge.insertElementAt(start_waypoint, Start_v);
        Edge.insertElementAt(end_waypoint, End_v);
        Edge.insertElementAt(road_cost, Score);
        Edge.insertElementAt(edge_direction, Direction);
        return Edge;
}


/*** score_Road: get the score (function of threat level and distance) of Road ***/


Real score_Road (RoadT Road)
{
        Real D = get_Road_Length (Road) / get_Total_Length();
        Real T = 0;
        Vector Threat_Map_Elem;
        Vector Position;
        Real Defeat;
        Real Bribe;
        String T_Type; String T_Size;
        Integer i = 0;

        while (i < Threat_Map.size()) {

                Threat_Map_Elem = Threat_Map.elementAt(i);

                Position = Threat_Map_Elem.elementAt(threat_position);

                if (Position.elementAt (road_id) == Road) {

                        T_Type = Threat_Map_Elem.elementAt (threat_type);

                        T_Size = Threat_Map_Elem.elementAt (threat_size);

                        Defeat = get_Defeatability (T_Type, T_Size);

                        Bribe = get_Bribability (T_Type, T_Size);

                        T = T *  (1 – Bribe) * (1 – Defeat);

                }

                i++;

        }

        return (Wd*D + Wt*(1 – T));

}


/*** score_Road_Section: return the score of a road section.                    ***/

/*** If Ref is 1, score the section from Road_Pos to start_waypoint ***/

/*** If Ref is 2, score the section from Road_Pos to end_waypoint ***/


Real score_Road_Section (RoadT Road, Real Road_Pos, Int Ref)
```

```
{
        // Find D
        Real D;
        if (Ref == 1) D.setValue (Road_Pos / get_Total_Length());
        Else D.setValue ((get_Length_Road (Road) – Road_Pos) / get_Total_Length());


        Real Defeat;
        Real Bribe;
        String T_Type; String T_Size;


        // Find T
        Real T = 0;
        Vector Position;
        Integer i = 0;
        while (i < Threat_Map.size()) {
                Threat_Map_Elem = Threat_Map.elementAt(i);
                Position = Threat_Map_Elem.elemenAt(threat_position);
                if (Position.elementAt(road_id) == Road) {
                        if (((Position.elementAt(road_position) < Road_Pos) && (Ref == 1)) ||
                           ((Position.elementAt(road_position) >= Road_Pos) and (Ref == 2))) {
                        Defeat = get_Defetability (T_Type, T_Size);
                        Bribe = get_Bribability (T_Type, T_Size);
```

```
                    T = T + Bribe + (1 – Bribe) * Defeat;
                    }
          }
          i++;
     }
     return (Wd*D + Wt*(1 – T));
}


/*** get_Road_Length: Look up Terrain_Map, return the length of Road ***/


Real get_Road_Length (RoadT Road)
{
     Vector Terrain_Map_Elem;


     Integer i = 0;
     while (i < Terrain_Map.size()) {
          Terrain_Map_Elem = Terrain_Map.elementAt(i);
          if (Terrain_Map_Elem.elementAt (road_id) == Road)
                    return Terrain_Map_Elem.elementAt (road_length);
          i++;
     }
}
```

```
/*** get_Total_Length: look up Terrain_Map and get the sum of the lenghts of all roads ***/


Real get_Total_Length (RoadT Road)
{
        Vector Terrain_Map_Elem;


        Real Len = 0;
        Integer i = 0;
        while (i < Terrain_Map.size()) {
                Terrain_Map_Elem = Terrain_Map.elementAt(i);
                Len = Len + Terrain_Map_Elem.elementAt (road_length);
                i++;
        }
        return Len;
}


/*** get_ Defeatability: look up Threat_Risk_Table and return the defeatability given type and
size of threat ***/


Real get_Defeatability (ThreatT T_Type, String T_Size)
{
        Vector Table_Elem;
        Integer = 0;

        while (i < Threat_Risk_Table.size()) {

                Table_Elem = Threat_Risk_Table.elementAt (i);

                if ((Table_Elem.elementAt (t_type) == T_Type) && (Table_Elem.elementAt
(t_size))

                        return Table_Element.elementAt (defeat);

        }

}
```

/*** get_ Bribability: look up Threat_Risk_Table and return the bribability given type and size of threat ***/

```
Real get_Bribability (ThreatT T_Type, String T_Size)
{

        Vector Table_Elem;


        Integer = 0;

        while (i < Threat_Risk_Table.size()) {

                Table_Elem = Threat_Risk_Table.elementAt (i);

                if ((Table_Elem.elementAt (t_type) == T_Type) && (Table_Elem.elementAt
(t_size))

                        return Table_Element.elementAt (bribe);
        }
}


/*** FUNCTIONS IN OTHER LIBRARIES ****/

Vector shortest_Route_Dijikstra (String Start_v, String End_v, set Vertices, set Edges)
{
        // Use Dijikstra algorithm to return the shortest route from start_v to end_v
        // Parameters:
        // - start_v and end_v: starting and ending vertices (waypoints)
        // - Vertices: Waypoints_List + {“start_v”, “end_v”}
        // - Edges: an edge is a directed arc and we represent as a pair (road_id, edge_direction)
        // - Cost: the cost are expressed in the Edges
        // The Dijikstra algorithm returns a sequence of edges that represents the shortest path
        // from vertex start_v to vertex end_v
}
```

/*** PROCEDURES KEY FIGURE MOTION MANAGER ****/***

```
/*** route_Distance: get the distance between Position1 and Position2 by following Route ***/

Real route_Distance (Real Position1, Real Position2, set Route)
```

```
{
        RoadT Road_1 = Position1.elementAt (road_id);
        RoadT Road_2 = Position2.elementAt (road_id);
        Real Road_Pos1 = Position1.elementAt (road_position);
        Real Road_Pos2 = Position2.elementAt (road_position);
        Real Len1 = get_Road_Length(Road_1);
        Real Len2 = get_Road_Length(Road_2);

        // if pos1 and pos2 are in same road, return their difference of their positions (absolute value)
        if (Road_1 == Road_2) return abs (Road_Pos1 – Road_Pos2);

        // if they are not in same road, sum the distances for each road
        Real Distance = 0;

        // Sum the distance from Position1 to one of Road_1 vertices
        Vector Route_Elem = Route.elementAt(0);
        if (Route_Elem.elementAt (road_direction) == “forward”) Distance = Distance + Road_Pos1;
        else if (Route_Elem.elementAt (road_direction) == “backward”) Distance = Distance + (Len1 – Road_Pos1);
```

```
    // Sum the distance from Position2 to one of Road_2 vertices
    Route_Elem = Route.elementAt(Route.size() – 1);
    if (Route_Elem.elementAt (road_direction) == “forward”) Distance = Distance + Road_Pos2;
    else if (Route_Elem.elementAt (road_direction) == “backward”) Distance = Distance + (len2 – road_pos2);


    RoadT Road;
    // Sum the lengths of the remaining roads
    Integer i = 1;
    while (i < Route.size() – 2) {            // Note: if pos1 and pos2 are not in same road, then route size is at least 2
        Route_Elem = Route.elementAt(i);
        Road = Route_Elem.elementAt(road_id);
        Distance = Distance + get_Road_Length (Road);
    }
    return Distance;
}


/*** check if Position is a safe house position  ****/

Boolean is_Safe_House_Position (set Position)
{
        String Safe_House;

        Vector Safe_House_Positions;

        Vector SH_Coord;

        Integer i = 0;


        while (i < Safe_Houses_List.size()) {

                Safe_House = Safe_House_List.elementAt(i);

                Safe_House_Positions = get_Safe_House_Coordinates (Safe_House);

                j = 0;

                while (j < Safe_House_Positions.size()) {

                        SH_Coord = Safe_House_Position.elementAt(j);

                        if (((SH_Coord.elementAt(road_id) == Position.elementAt(road_id)) &&

                           (SH_Coord.elementAt(road_position)                                ==
Position.elementAt(road_position))) return true;

                }

                i++;

        }

        return false;

}


/*** get_Safe_House_Coordinates: given a Safe_House, return a list of possible ways to express
its position ***/
```

Vector get_Safe_House_Coordinates (String Safe_House)

```
{
        Integer i = 0;
        Vector Safe_House_Position;
        Vector Position_List;

        while (i < Terrain_Map.size()) {

                // check if safe house is a start_waypoint of a road --> position is (road, 0)
                Terrain_Map_Elem = Terrain_Map.elementAt(i);
                if (Terrain_Map_Elem.elementAt(start_waypoint) == Safe_House) {
                        Safe_House_Position.insertElementAt (road_id,
Terra_Map_Elem.elementAt(road_id));
                        Safe_House_Position.insertElementAt (road_position, 0);

                        Position_List.addElement (Safe_House_Position);
                }

                // check if safe house is a end_waypoint of a road --> position is (road, road
length)
                if (Terrain_Map_Elem.elementAt(end_waypoint) == safe_House) {
                    Safe_House_Position.insertElementAt (road_id,
Terrain_Map_Elem.elementAt(road_id);

                    Safe_House_Position.insertElementAt (road_position,
Terrain_Map_Elem.elementAt(road_length));


                    Position_List.addElement (Safe_House_Position);

                }

                i++;

        }

        return Position_List;

}


/*** find_Next_Position: Given a distance, a position, and a route, the function returns the next
position ***/


Vector find_Next_Position (Real Distance, Vector Position, Vector Route)
{

        Real D = Distance;

        Vector Cur_Pos = Position.copy ();

        Vector Cur_Route = Route.copy ();

        Vector Cur_Route_Elem, Next_Route_Elem;


        RoadT Cur_Road;
```

```
    Real Cur_Road_Pos;
    String Cur_Direction;
    Real Len;

    while (D > 0) {

        Cur_Route_Elem = Cur_Route.elementAt (0);
        Cur_Direction = Cur_Route_Elem.elementAt (road_direction);
        Cur_Road = Cur_Pos.elementAt (road_id);
        Cur_Road_Pos = Cur_Pos.elementAt (road_position);

        if (Cur_Direction == “forward”) {
            Len = get_Road_Length (Cur_Road));
            if ((Cur_Road_Pos + D) > Len) {
                D = Cur_Road_Pos + D – Len;          // remaining D after
reaching vertex

                // Move to next coordinate, update Cur_Route and Cur_pos
                Next_Route_Elem = Cur_Route.elementAt (1);
                Cur_Pos.setElementAt (road_id, Next_Route_Elem.elementAt
(road_id));
```

```
                    Cur_Pos.setElementAt   (road_position,   get_Road_Start
(Next_Route_Elem));

                    Cur_Route.removeElementAt(0);


                }

                Else {

                    D = 0;

                    Cur_Pos.setElementAt (road_position, Cur_Road_Pos + D);

                }

            }

            Else if (Direction == “backward”) {

                if ((D > Cur_Road_Pos) {

                    D = D – Cur_Road_Pos;


                    // Move to next coordinate, update Cur_Route and Cur_pos

                    Next_Route_Elem = Cur_Route.elementAt (1);

                    Cur_Pos.setElementAt   (road_id,   Next_Route_Elem.elementAt
(road_id));

                    Cur_Pos.setElementAt   (road_position,   get_Road_Start
(Next_Route_Elem));

                    Cur_Route.removeElementAt(0);
                }
                Else {
                        D = 0;
                        Cur_Pos.setElementAt (road_position, Cur_Road_Pos + D);
                }
        }
    }

    return Cur_Pos;
}
```

/*** get_Road_Start: given a route element, return 0 if direction is forward or the route length if backward ***/

Real get_Road_Start (Vector RouteElem)

```
{
    if (RouteElem.elementAt (road_direction) == "forward") return 0;
    else if (RouteElem.elementAt (road_direction)  == "backward")
        return get_Road_Length (RouteElem.elementAt (road_id));
}
```

void update_Truth_Table_Position (String Actor, Vector Position)

```
{
        Vector Table_Elem;
        Vector Position;


        RoadT Road = Position.elementAt (road_id);
        Real Road_Pos = Position.elementAt (road_pos);


        i = 0;
        while (i < Ground_Truth_Table.size()) {
                Table_Elem = Ground_Truth_Table.elementAt (i);
                if (Table_Elem.elementAt (actor_name) == Actor) {
                        Position = Table_Elem.elementAt (actor_position);
                        Position.setElementAt (road_id, Road);
                        Position.setElementAt (road_position, Road_Pos);
                        Table_Elem.setElementAt (actor_position, Position);
                        Ground_Truth_Table.setElementAt (i, Table_Elem);
                        break;
                }
        }
}
```

/** PROCEDURES THREAT ANALYZER *****/***

```
void process_A_Message (Vector Message, Vector T_Map)
{
        i = 0;
        while (i < T_Map.size) {
                T_Map_Elem = T_Map.elementAt(i);
                if (Message.elementAt (threat_id) == T_Map_Elem.elementAt (threat_id)) {
                        T_Map_Elem.elementAt (threat_position) = Message.elementAt (threat_position);
                        T_Map_Elem.elementAt (position_range) = 0;
                        T_Map.setElementAt (i, T_Map_Elem);
                        break;
                }
                i++;
        }
}
```

/* PROCEDURES THREAT SENSOR ****/***

```
Real distance (Vector Position1, Vector Position2)
{
        Vector Route = get_Shortest_Route (Position1, Position2);

        return route_Distance (Position1, Position2, Route);

}


void send_Message (ThreatT Threat, Vector Position)

{

        Vector Message;

        Message.insertElementAt (threat_id, Threat);

        Message.insertElementAt (threat_position, Position);



        Threat_Sensor_Data.addElement (Message);

}


Vector get_Shortest_Route (Vector Position1, Vector Position2)

{

        RoadT Road_1 = Position1.elementAt (road_id);

        RoadT Road_2 = Position2.elementAt (road_id);

        Real Road_Pos1 = Position1.elementAt(road_position);

        Real Road_Pos2 = Position2.elementAt(road_position);

        Real Road_Len1 = get_Road_Length(Road_1);

        Real Road_Len1 = get_Road_Length(Road_1);



        Vector Route_Elem;
```

```
    // check if pos1 and pos2 are in the same road, if yes, the route is the
    if (Road_1 == Road_2) {

            Route_Elem.insertElementAt (road_id, Road_1);
            if (Road_Pos2 >= RoadPos1) Route_Elem.insertElementAt (road_direction,
“forward”);
            Else Route_Elem.insertElementAt (road_direction, “backward”);

            Route.addElement (Route_Elem);
            return Route;
    }

    // if they are not in same road, get the 4 vertices
    String S1 = startWp (Road_1);
    String E1 = endWp (Road_1);
    String S2 = startWp (Road_2);
    String E2 = endWp (Road_2);

    // Buid Vertices and insert two more vertices corresponding to pos1 and pos2
    Vector Vertices = Waypoints_List.copy ();
    Vertices.addElement (“start_v”);
    Vertices.addElement (“end_v”);


    // Build Edges, insert new edges for Position1 and Position2
    Edges.addElement (New_Edge (r1a, S1, “start_v”, Road_Pos1, “forward”);
    Edges.addElement (New_Edge (r1a, ”start_v”, S1, Road_Pos1, “backward”);
    Edges.addElement (New_Edge (r1b, E1, “start_v”, Road_Len1 – Road_Pos1, “forward”);
    Edges.addElement (New_Edge (r1b, ”start_v”, E1, Road_Len1 – Road_Pos1, “backward”);
    Edges.addElement (New_Edge (r2a, S2, “end_v”, Road_Pos2, “forward”);
    Edges.addElement (New_Edge (r2a, ”end_v”, S2, Road_Pos2, “backward”);
    Edges.addElement (New_Edge (r2b, E2, “end_v”, Road_Len2 – Road_Pos2, “forward”);
    Edges.addElement (New_Edge (r2b, ”end_v”, E2, Road_Len2 – Road_Pos2, “backward”);


    Vector Route = shortest_Route_Dijisktra (“start_v”, “end_v”, Vertices, Edges);


    // Replace r1a = road_1, r1b = road_1, r2a = road_2, r2b = road_2
    Integer i = 0;
    while (i < Route.size()) {
        Route_Elem = Route.elementAt(i);
```

```
                    if ((Route_Elem.elementAt(road_id) == r1a) || (Route_Elem.elementAt(road_id)
== r1b)) {
                              Route_Elem.setElementAt (road_id, Road_1);
                              Route.setElementAt (i, Route_Elem);
                    }
                    if ((Route_Elem.elementAt(road_id) == r2a) or (Route_Elem.elementAt(road_id)
== r2b)) {
                              Route_Elem.setElementAt (road_id, Road_2);
                              Route.setElementAt (i, Route_Elem);
                    }
                    i++;
          }
          return Route;
}
```

Boolean random_Binary (Real Success_prob)

```
{
          if (u() <= Success_prob) return “true”;
          Else return “false”;
}
```

/*** FUNCTIONS DEFINED IN OTHER PLACES ***/

// In Key Figure Route Planner

Real get_Road_Length (RoadT Road) {}
Vector shortest_Route_Dijikstra (String Start_v, String End_v, set Vertices, set Edges) {}

/*** FUNCTIONS IN OTHER LIBRARIES ****/

Real u()

{

// Random variable with uniform distribution: return a number between 0 and 1 with uniform distribution

// Look algorithms or library for random number generation

}

/ PROCEDURES THREAT PHONE CALLER ****/***

/*** FUNCTIONS DEFINED IN OTHER PLACES ***/

// In Threat Sensor

Boolean random_Binary (Real Success_prob) {}

void send_Message (ThreatT Threat, Vector Position) {}

*/*** PROCEDURES GOOD GUYS ROUTE PLANNER ***/***

/** lost_Target_Position lost the position of the target if the range includes the vertices of the road **/

```
Boolean is_Position_Accurate (Real Pos_Range)
{
        RoadT Road = Target_Position_Estimate.elementAt (road_id);
        Real Road_Pos = Target_Position_Estimate.elementAt (road_position);
        Real Road_Len = get_Road_Length (Road);

        return ((Target_Position_Range <= Road_Pos) && (Target_Position_Range <=
(Road_Len – Road_Pos));
}
```

/** is_Target_Captured checks if the position of the distance to target is within the capture range **/

```
Boolean is_Target_Captured (Vector Position)
{
        Targ_Position = get_Truth_Table_Position ("key figure");

        Real D = distance (Position, Targ_Position);
        return (D <= Capture_Range);
}

/** get_Truth_Table_Position look up Ground_Truth_Table for the real position of Actor **/

Vector get_Truth_Table_Position (String Actor)
{
        Vector Table_Elem;

        Integer i = 0;
        while (i < Truth_Table_Position.size()) {
                Table _Elem = Ground_Truth_Table.elementAt(i);
                if (Table_Elem.elementAt (actor_name) == Actor) return
Table_Element.elementAt(actor_position)
                i++;
        }
}

/*** FUNCTIONS DEFINED IN OTHER MODULES ***/

// Threat Sensor
Vector get_Shortest_Route (Vector Position1, Vector Position2)
```

```
// In Key Figure Route Planner
Real get_Road_Length (RoadT Road) {}




/*** FUNCTIONS IN OTHER LIBRARIES ****/


Real rand_Int (Int a, Int b)
{
        // Random variable that generates a random integer between the interval [a, b]
        // e.g. rand_Int (0, 4) returns, with equal probability, one of the numbers: 0, 1, 2, 3, 4.
}
```

/ PROCEDURE GOOD GUYS MOTION MANAGER ****/***

```
/*** FUNCTIONS DEFINED IN OTHER MODULES ***/


// Key Figure Motion Manager
Real route_Distance (Real Position1, Real Position2, set Route) {}
Vector find_Next_Position (Real Distance, Vector Position, Vector Route) {}
void update_Truth_Table_Position (String Actor, Vector Position) {}


// Threat Sensor
Real distance (Vector Position1, Vector Position2) {}
```

/ PROCEDURES SENSOR READER ***/***

```
void decay (Vector Position_Range)
{
        Target_Position_Range = Target_Position_Range + Decay_Rate;
}
```

/ PROCEDURES TARGET SENSOR ****/***

```
/*** FUNCTIONS DEFINED IN OTHER MODULES ***/


// Key Figure Motion Manager
Boolean is_Safe_House_Position (set Position) {}


// Threat Sensor
Real distance (Vector Position1, Vector Position2) {}
Boolean random_Binary (Real Success_prob) {}


// Good Guys Motion Manager
Vector get_Truth_Table_Position (String Actor) {}
```

/ PROCEDURES TARGET PHONE CALLER ****/***

```
/*** FUNCTIONS DEFINED IN OTHER MODULES ***/
```

```
// Threat Sensor
Real distance (Vector Position1, Vector Position2) {}

Boolean random_Binary (Real Success_prob) {}
```