

TRANSIMS Research and DEPLOYMENT Support

-

Draft of Quarterly Progress Report

Introduction

Position of FHWA

Through a series of discussions with project sponsors, we have developed an understanding of the role and needs of the FHWA, the Office of Planning, and the Systems Planning and Analysis Team. As stated by Jeremy Raw in his white paper ““Next-Generation” Regional Simulation Meetings”:

The role of FHWA, the Office of Planning, and the Systems Planning and Analysis Team supports practical deployment of advanced techniques that can be adopted and used by organizations involved in transportation planning activities. The guiding principle of such research and development is to selectively repackage results from the academic search for comprehensive truths into tools that provide effective decision support.

Based on these meetings, we observed the following core capability shortcomings that FHWA finds with the currently existing transportation models. Traditional transportation models are inflexible in the sense that they are not readily adaptable for certain spheres of evaluation, particularly in the area of new or esoteric transportation technologies and management strategies. These models are designed to be used for fixed purposes and thus only able to address the system elements developers deemed important upon software conception. This is the impetus for the project to address the modeling of ITS infrastructure.

Transportation models are insularly developed and are not built to interoperate easily with each other. When transportation planners approach a problem, they may need to utilize a suite of tools in order to answer the necessary questions across all model scopes. However, due to this lack of federation among tools, it is either expensive to do so or perhaps even infeasible. This is the primary motivation for the project to address the creation and formalization of an interoperability protocol between this and other models.

Transportation models do not properly support the core decision making process when they are applied by end-users. They tend to elucidate one possible system outcome with one set of performance measures. However, that is often insufficient to address the range of outcomes which would be necessary to evaluate and validate a real world project, such as new infrastructure construction. This is the impetus for development in this project to keep the specific problem well in mind and to ensure the focus is on providing planners with the appropriate performance measures or allowing the planner to dictate the performance measure.

Transportation models are very often applied in the confines of a transportation planning project, as such it is vital for these tools to keep the transportation planning process scope strongly in mind.

Transportation planning projects are aimed at solving a specific problem in a given amount of project time. The average modeler is not always familiar with scripting and often does not have sufficient time or data to build a model with an extremely high level of detail if it is not relevant to the problem at hand. This the impetus for development in this project to keep the model development and execution process simple as well as to require only the minimum data which is sufficient to answer the question at hand.

TRACC's Observations

The needs of the transportation planning community are extremely varied and change over time as the modern transportation system becomes more complex with advances in technology. In addition, each planning group has different needs, as each must solve a slightly different planning problem or a similar planning problem on different transportation systems (one group may be concerned with regional emissions whereas another will be concerned with testing charging stations). Owing to these two constraints and historical evidence from the efficacy of comprehensive models such as TRANSIMS, it is apparent that developing an all-encompassing "mega" model, which is inherently inflexible, is not a tenable approach. It is unlikely developers would be able to conceive of every possible need the software must address using a minimalistic set of data at the software creation stage.

Despite having a large body of research, there is proportionally little work being done to unify the modeling concepts produced by the transportation research community. Research is often conducted on specific aspects of the transportation system without an eye for how it fits into the larger picture. For example, a new car following simulation routine might be developed which is fast and effective, however there is no way for the routine to be "slotted in" with all other components necessary such that its merits can be realized (or perhaps challenged) on a system-wide scale. Unification is brought about largely through the efforts of private tool developers such as PTV, Quadstone, and Caliper who do not necessarily hold the same set of motivations as an open research community. The Transportation Research Board's Strategic Highway Research Program (SHRP 2) has funded several research projects (e.g. C04, C10, and L04) to develop methodologies to integrate user response and travel demand models in network simulation procedures in light of existing demand and network modeling tools. However, these efforts are still not in a fully integrated modeling environment.

The concept of simulation remains a potent tool in the transportation planning arsenal. A simulation allows a modeler to create a virtual representation of a complex system and observe how it behaves, ultimately enabling the succinct summarization of system-wide behavior. Traditionally in a transportation 4 step model, simulation is conceived of as a "one-shot" tool - given these set of inputs, "show me what happens in this given day". This approach is generally understood to be invalid in isolation, the main question being, "does the system represent a 'normal day' (most common) case?" This is typically ensured by performing an iteration loop to stabilize all "actions" of the transportation system members. Once it is reasonably well assumed that this case is met, it is determined that the simulation is a reasonable analog for reality. Performance measures are drafted off of this model (or case studies involving this model) and synthesized with other criteria to make a decision. However, this procedure is notably destitute of describing the inherent variation in the system - a highway might be determined to be optimal for the "normal day", but under different weather conditions or days of the week there may have been a far better alternative. Therefore, the typical model of "one shot" transportation needs to evolve to be able to describe this variation.

The technological utilization and awareness of advanced research within the transportation community

can be somewhat lower compared to others such as the agent-based modeling or artificial intelligence communities. Implementing many complex and high resolution techniques are met with skepticism as it is generally understood that a transportation system simulation has the potential to be very computationally intensive. However, computing continues to advance rapidly and performing ambitious simulations is not the challenge that it once was if the software is appropriately designed. In this same vein, there is a burgeoning of new data sources (highway blue tooth sensors, online wiki-style road networks, traffic social networks, etc.) which, if properly utilized, provide clarity and validation to the entire modeling process while expediting model construction. As transportation planning moves steadily into the realm of utilizing simulation-based modeling, it begins to hold more and more in common with other simulation heavy fields such as the artificial intelligence and agent-based modeling communities. Thus, it becoming increasingly important to review and apply the literature and techniques of these related fields.

TRACC's Conceptualization of the Ideal Tool for FHWA

Based on the above needs and comments, TRACC has conceptualized in this section an "ideal tool" which is not attainable at the present, but will hopefully serve as a guide for how to take development steps in the right direction. In the ideal case, what is needed is an adaptable transportation model that can be used in a variety of contexts and with various types of available data - one which can be supported easily and connected with other major efforts in the transportation modeling community. In the hands of a typical planner - the model should allow them to describe the type of problem they are trying to solve, the important performance measures needed to assess success, and feed in relevant data they have easily available. The model should run quickly and allow the planner to examine the results allowing them to review outputs targeted at the performance measures and the core decision which needs to be made. If the modeler has other tools at their disposal, they should be able to run them and easily integrate their results to enhance the decision making process such that it becomes more comprehensive and better informed.

In the hands of an advanced planner interested in evaluating a new technology or system - the model should allow them to describe the new technology along with the other transportation system elements they wish to examine its effects on. Once run, the model should allow the planner to examine the results from many angles and easily add or remove contingent systems to get a sense for what the change in system behavior is when the new technology is utilized and how it works in practice. Their results could be easily shared with all other software users to give them insight into their own planning process as well as reveal the efficacy of new technologies.

In the hands of a transportation researcher - the model should allow them to test out their new theories by easily describing the altered agent or system behavior as well as measures of interest. Once run, the model would demonstrate how the overall system changes with the altered behavior and thus give the researcher a sense for how their theory works in practice and ways it could be improved. They would be able to easily re-arrange the routines until the theory is developed to the point they desire. This advancement then could be easily shared with all users so they might make use of the improved routine.

TRACC's Strategy - Take Foundational Steps Toward the Ideal Tool

A logical approach to tackle the problem is to develop a unifying transportation model development framework. This is a platform which allows for an easy and flexible way to describe and execute most potential transportation models. In order to illustrate this concept, we will refer to the approach generally followed by the agent-based modeling community.

In essence, an agent-based modeling package allows one to apply what might be called a “modeling language” (notably distinct from a programming language) to describe the model at hand. In graphical versions of an agent based software, the user might drag and drop shapes onto the screen to build a flow chart of behaviors. In the case of Repast Flowchart, a block represents a function, diamond is a condition, ellipse is a goal or initial condition, and so forth. The underlying basis of the graphical tools is generally a written modeling language. In the case of Repast, the underlying language is called ReLogo; upon first glance, it initially appears like a programming language, however when the syntax is more closely examined one can see that it actually bears more resemblance to what was described in the flowchart version. The focus is on describing agent behaviors, so the language provides common functions such as “getXCor()” (retrieving the x coordinate of the agent) or “step()” (indicating when time steps forward, what the agent does) or “ask(type,closure)” (indicating a communication action with some other agent of a given type). In both cases, once the model is constructed it can also be run (with no additional input from the modeler) within the same framework to determine the results.

Agent-based models of this kind are widely used to describe and evaluate an incredibly diverse set of systems. The analogy can be made that as a programming language such as C++ can be used to make any software program, a modeling language such as ReLogo can be used to make any agent-based model. Thus, these models are utilized by communities much the same way as the “Ideal Tool for FHWA” described above. Given a real world system model-able in agent-based language, the user is able to use helpful language to describe the aspects of the system which are of interest, culminating in a model. When this model is supplied with a virtualization of the real world information, it is able to execute and simulate the system. Upon termination, the user can extract the performance measures they deemed as being of interest. Perhaps, most importantly, the model is fundamentally compatible with any other model written in this “modeling language” and anyone who is familiar with it can pick it up and make alterations or add components.

It is the assertion of TRACC that a transportation model is representable as an agent-based model and thus simply a special class of it. It is understood that it is infeasible for the transportation community to build from the ground up using a language like ReLogo - because this language is too low level and designed to be able to describe any possible agent-based model model. We believe that past model unification efforts such as TRANSIMS, TransCAD, MATSIM, etc., has revealed that transportation system models generally have a great deal of commonality among them and cover a relatively finite conceptual space. Concepts which would be too complex to describe in the more low level space of a language like ReLogo may be considered fundamental in transportation models. For instance, transportation models are typically very concerned with spatial movement (particularly through networks): creating paths through it, moving through it, and analyzing it. In another example, transportation models typically have extremely complex planning agents (human travelers): deciding on an activity schedule,

choosing a destination, choosing a mode of travel, and so forth. In fact, modeling these same types of agents (human travelers) at some level of resolution is a common point in the majority of commercial transportation modeling packages.

Therefore, we believe that the appropriate development would be:

- A **high level** “agent-based” modeling **language** targeted **specifically** at transportation - a minimalistic set of concepts, building blocks, and symbols which can be utilized to succinctly describe all possible elements of a transportation system
- A **framework** which facilitates the **development, execution, and review** of a model **written** in such a language
- A **case study** demonstrating the **application** of this language and framework to a problem which is deemed currently **difficult to evaluate** by FHWA such as a **Command and Control Center or other ITS system**
- A **case study** demonstrating **how** a model written in this language and framework can **interoperate** with an **existing** transportation tool which is deemed as **important** by FHWA
- A **conceptualization** of the **path forward** for such an approach and **next steps** to begin **transferring** the technology to the planning community

An Illustrative Example of Transportation Modeling Language Application

Consider the case of a timed signal and a car traveler. Now, consider a specific behavior of these two entities such as the signal choosing its’ phasing plan vs. the car traveler choosing its’ route to a destination. In a model such as TRANSIMS, these two behaviors are modeled in an entirely different (and very specialized) manner, they have nothing to do with one another nor are their execution or storage processes at all related. Alternatively, they could be “symbolized” as two *agents* invoking two different *planning actions* which results in a schedule of intended actions (or *plan*) – say a “Planning Agent”. So, instead of modeling them as two completely separate entities, they could share common parentage and become inherently more relatable to one another and interoperable. Now, say that the planning decision at hand is to evaluate the efficacy of a new “signal voting system” where drivers submit their intended routes to a regional server which consolidates this information and changes the light timings. When the “transportation modeling language” is used to describe the model, the signal already inherently understands what a “Planning Agent” looks like (it is itself also a “Planning Agent”) and thus knows how it can ask the travelers for information, what format it expects to receive it in, and how to synchronize this interchange so there is no data corruption.

Anticipated Benefits of the Transportation Modeling Language and Framework Approach

It is the belief of TRACC that constructing a common modeling language for transportation systems and providing a framework to apply it will positively impact many of the areas which were described as transportation model inadequacies:

The language encourages research projects to collaboratively apply concepts on a broader scope and easily test theories by building an open source “library” of model “fragments”. By being able to implement, say, a new routing or signal phasing algorithm (a model “fragment”) easily in a more comprehensive transportation model already built using the framework (coming from this open source “library”) - the researcher is able to assess their theories without being concerned about also needing to re-implement all of the other transportation modeling components (such as discrete event execution or micro simulation movements). Further, it opens the possibility for research communities to more easily share model “fragments” among themselves and cooperatively assemble them into a more comprehensive model.

The language inherently encourages interoperability. By being able to describe another model in terms of this language, it becomes much more easy to interact with it, likely only a translation routine would need to be written. For instance, the framework could be used to create an activity based model which depends on the current location of an individual on the road to determine the departure time of its’ next activity. This model would understand that the spatial information of a traveler means symbolically the same thing whether it comes from an internal micro simulation model or from a file (after translation) which is output from another model such as CORSIM – the only difference would be in the level and type of coordination allowed between the activity planner and micro simulator.

The language would allow the construction of models which target the problem at hand. If the modeler does not believe it is necessary to have a microscopic level of detail (for instance, they do not have exact lane data), they can remove that component or replace it with a library of other lower resolution routines. If the modeler is interested in a specialized performance measure, they would be able to develop that custom measure specifically for their model and attach a “virtual observer” to watch how that measure develops over time.

Finally, the language encourages a modeling community which keeps the planning process in mind. As suggested above, it would be possible for the research community to maintain an open source “library” of different models written using the language and framework. A planner would be able to select the model which is most appropriate (in scope and problem target) for his or her application and then get to work immediately, they would be able to make minor high level customizations to the model to suit it precisely to their needs.

Model Development and Design

Naming

Currently we are considering two major variations for the name of the framework to be developed. For the purposes of clarity, the name POLARIS will be used for the remainder of this report to refer to the model.

1. **POLARIS (Planning and Operations Language for Agent-based Regional Integrated Simulation) Transportation modeling framework**

2. InSiGHT (Integrated Simulation of Generalized Human Transportation)

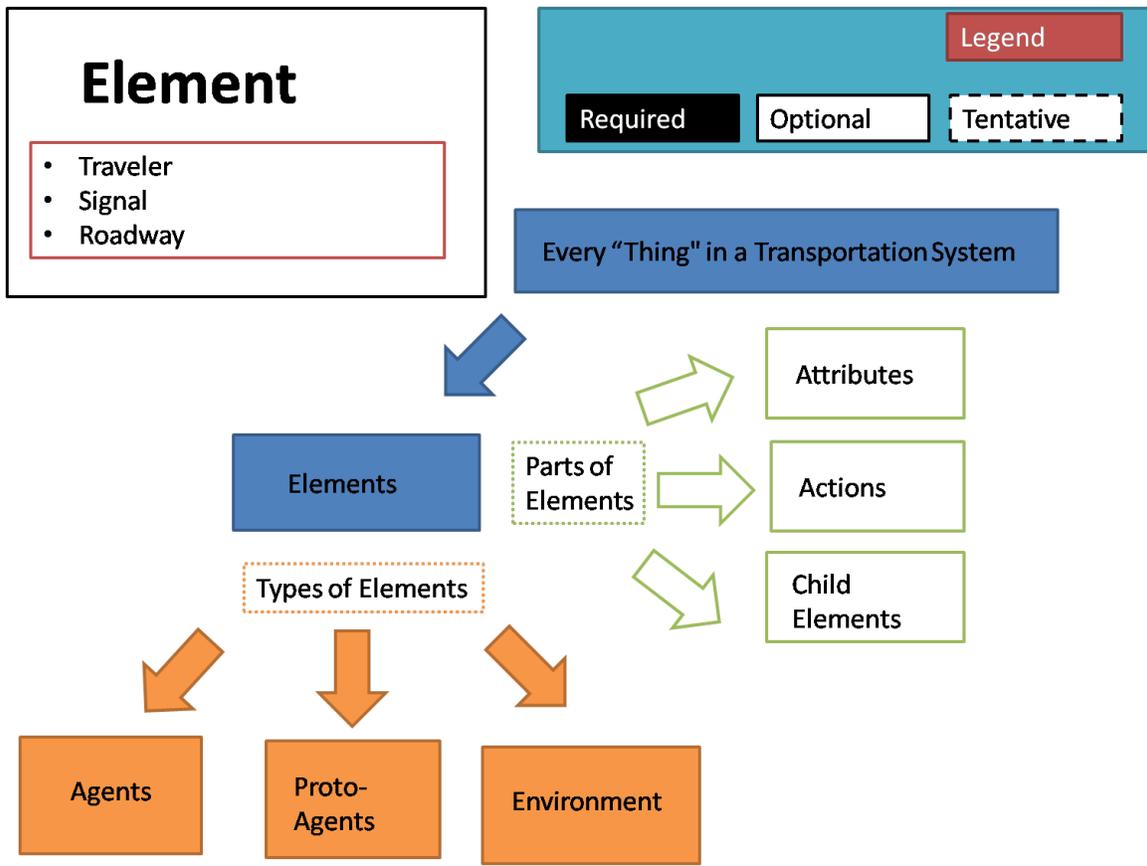
POLARIS Development Introduction

As stated in the introduction, the current strategy being undertaken to solve the issues laid out by FHWA is to develop a transportation modeling language, a model construction and execution framework, and case studies which illustrate how these may be used to provide interoperability and describe ITS systems.

The first step in the project is to clearly identify the representation of “language” to be used throughout the development cycle. By language we mean key notations and concepts to be used to describe the complex model of a transportation system. Some parts will be defined using a rigorous mathematical formalism and some are more vaguely defined to allow flexibility. The logical basis of this language is that of the agent-based modeling community, therefore many of the terms seen in our transportation modeling language have similar meaning in an agent-based model. The agent-based modeling approach is a relatively new concept and thus the language used in the literature by different authors varies. Thus, it is important to clarify the definitions of all key concepts which may have overlapping usage in the agent-based modeling community.

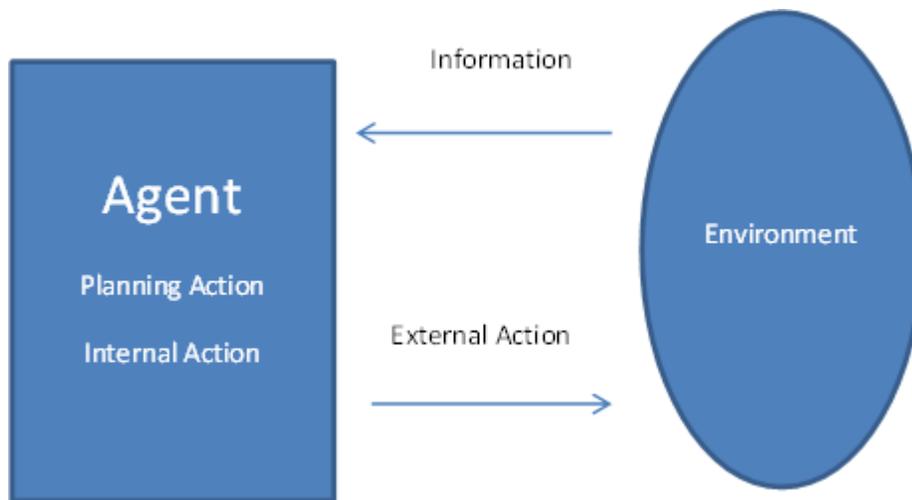
In this section, the focus will be on introducing how to re-frame transportation models in terms of agent-based language and concepts rather than on rigorous definition. The section “POLARIS Core Categories” will provide a more detailed explanation of the rationale behind each “word” chosen, where they fit in the hierarchy of other “words”, as well as examples of how they may be used.

The “things” in a traditional agent-based model fall into three major categories: agents, proto-agents, and environments. This is a clean way to group the “things” of a system roughly by the type and complexity of their functional behavior so it seemed reasonable to carry this set of categories over to POLARIS. In order to be able to refer to any “thing”, POLARIS uses the term “element” - agents, proto-agents, and environment are all different kinds of elements. This relationship is illustrated below.



The easiest way to think of the difference between each of these types of “elements” is through example. An agent are the more complex and dynamic constructs in a transportation model: people, intelligent traffic signals, households, and so on. The environment by contrast are the simple and passive constructs in the transportation model: roadways, rail lines, office buildings. The proto-agents are the category of things which are in-between the two: unintelligent traffic signals, gps devices, weather systems. Another word which will be used often and comes from the agent-based modeling community is “state.” This simply refers to a certain configuration of an element in the system. An example would be a traffic signal’s “color” at a given time.

We begin by discussing the core object - an agent. An agent is able to acquire and process information about the environment and then develop a plan to meet its’ needs. The plan is adapted to the information about the environment which is available to the agent. In fact, a plan is always representable as the set of actions which an agent may execute in order to achieve a goal state. Below is a depiction of the relationship between an agent and the environment in POLARIS.



We also would consider an agent which does not satisfy these minimum requirements to be called an intelligent agent. We would follow a notation suggested in (North & Macal, 2007) and would call such an agent a proto-agent. A proto-agent is not adaptive to an environment, i.e it does not adapt or change its behaviour based on the new information about the environment. Thus, it just executes actions according to some prescribed logic.

Environment:

Word environment in this context is overloaded. From the perspective of a system, the environment is all of the elements of the system which are not agents, i.e. infrastructure, resources, weather, etc. From the perspective of an agent, the environment is everything outside its internal state. Thus, to a given agent, other agents are essentially part of the environment as well.

Certain distinctions can be made between a **fully** vs. **partially observable** environment in terms of an agent. One could say that the environment is fully observable to an agent if the agent knows the entire state of the environment. In a partially observable environment, the agent only has information about part of the environment's state. To illustrate the distinction, consider that at an intersection, a traveler might be able to see the street ahead, but only have limited knowledge of the streets to the left or right.

In agent-based models further distinctions can be made between a **deterministic** and a **stochastic** environment. In a deterministic environment, the result of an action is "guaranteed". In a stochastic environment the result of an action can be guaranteed only with certain probability, less than 1.

Consider also information flow in relation to the environment; there is a distinction between information which is obtained by the agent actively looking for it (freely executing a percepts action) versus unexpected information which is "pushed" onto the agent.

Actions:

As identified above, we can distinguish several types of actions. There are those which modify an internal state of an agent (Internal Action) and one which affect the environment (External Actions). Among internal actions we can also distinguish a percepts action, action of acquiring information about the environment and planning action. In one definition of a planning action, the result is a developed plan (list of actions) which allows achieving the goal, or a message saying that such a plan does not exist,

provided actions available to the agent.

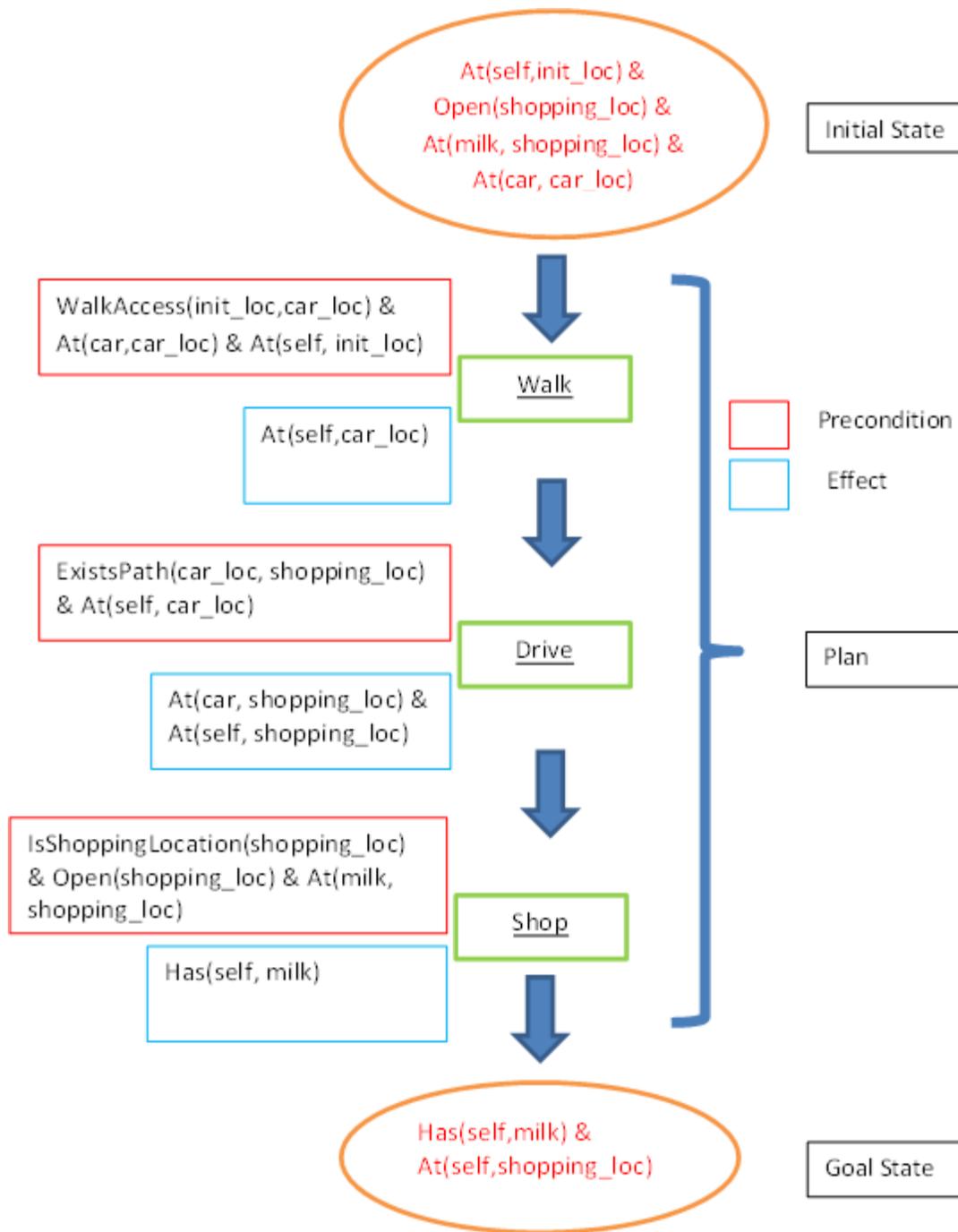
A set of actions available to an agent at a given state can be represented using classical planning notations (Russel & Norvig, 2010). In this book authors developed a concept of action schema which is used to represent agent's actions. Here is an example of an action schema for a shopping activity using a pseudo-code notation:

```
Action(Shop(product,loc),
      Precond:IsShoppingLocation(loc) & Open(loc) & At(product,loc)
      Effect: Has(product))
```

The schema has an action name, list of variables used, preconditions and an effect (or postcondition). The list of preconditions need to be satisfied in order for action to be applicable. Then, the the essence of an agent is to find a list of actions (plan) in order to satisfy it needs, provided an initial state of an agent. Here is an example of how an action schema can be used to model a milk shopping activity:

```
Init(At(self,init_loc) & Open(shopping_loc) & At(milk, shopping_loc) & At(car, car_loc)
Goal(Has(self,milk) & At(self,shopping_loc))
Action(Walk(self,init_loc, car_loc, car)
      Precond:WalkAccess(init_loc,car_loc) & At(car,car_loc) & At(self, init_loc)
      Effect: At(self,car_loc))
Action(Drive(self, car_loc,shopping_loc)
      Precond:ExistsPath(car_loc, shopping_loc) & At(self, car_loc)
      Effect: At(car, shopping_loc) & At(self, shopping_loc))
Action( Shop(milk, shopping_loc)
      Precond:IsShoppingLocation(shopping_loc) & Open(shopping_loc) & At(milk, shopping_loc)
      Effect: Has(self, milk))
```

Below is the same example in the form of a diagram:



Here we use a set of functions, which are defined as follows:

At(object,location) = true, if an object is located at location

Open(location) = true, if there are no institutional restrictions associated with location at the current time

Has(agent,object) = true, if agent possesses the object

WalkAccess(loc1,loc2) = true, if there is a walk path from loc1 to loc2

ExistsPath(loc1,loc2) = true, if there is a drive path from loc1 to loc2

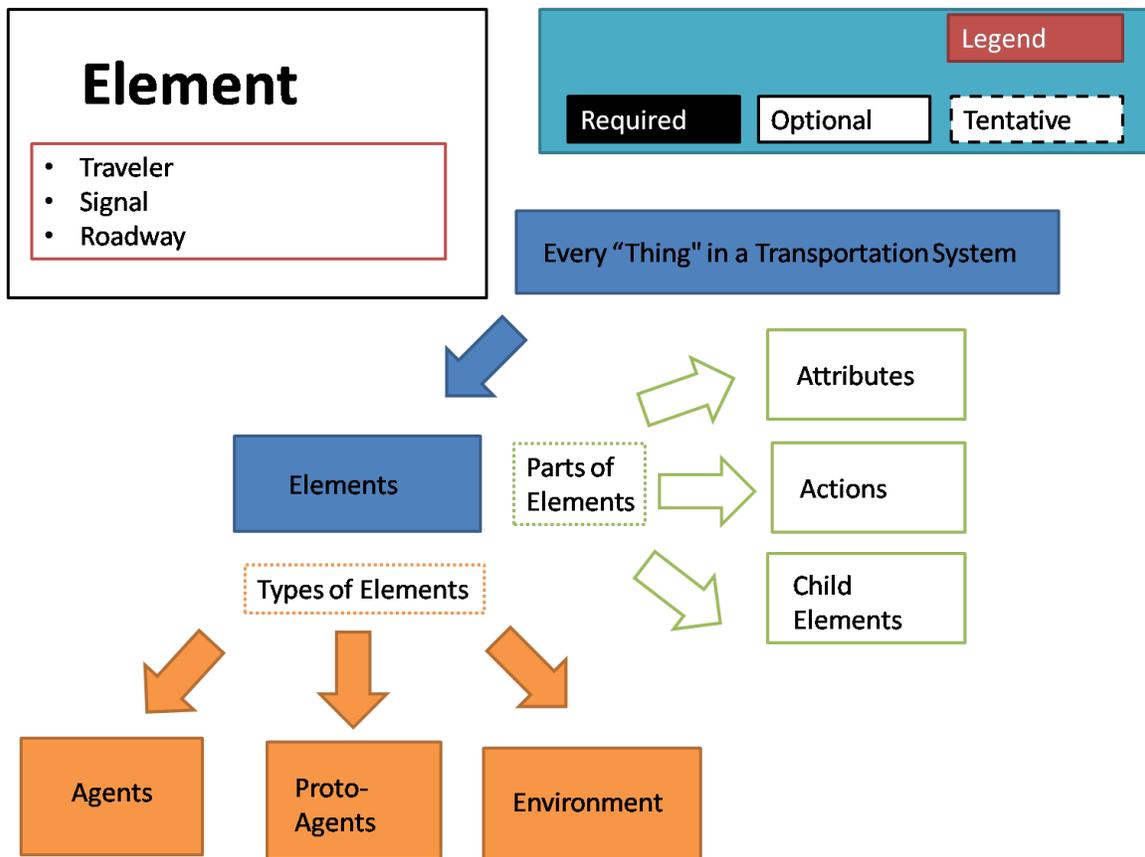
IsShoppingLocation(loc) = true, if the loc is a shopping location

The initial state and the goal defines pre- and post- conditions. Both are defined as a conjunction of atomic conditions. Then a planning action is used to develop a list of actions which are defined for an agent to satisfy all of the conditions identified in the Goal. Note, that an action is not necessarily an atomic unit which can be performed by an agent but can be a high level element, which might need to be refined in order to be executed by an agent. In the example above, the Drive action is not necessarily an atomic action depending on a traffic flow unit this action might need to be refined. If a traffic flow unit uses micro-simulator approach, then drive action need to be refined in terms of Accelerate, Decelerate, ChangeLane actions. This illustrates is a notion of a **hierarchical planning**.

These concepts, borrowed from the agent-based modeling community, have been used to generate our initial proposal for the design of the POLARIS modeling system. This proposed system design is described in detail in the section “POLARIS Modeling System - Initial Design”.

POLARIS Core Categories

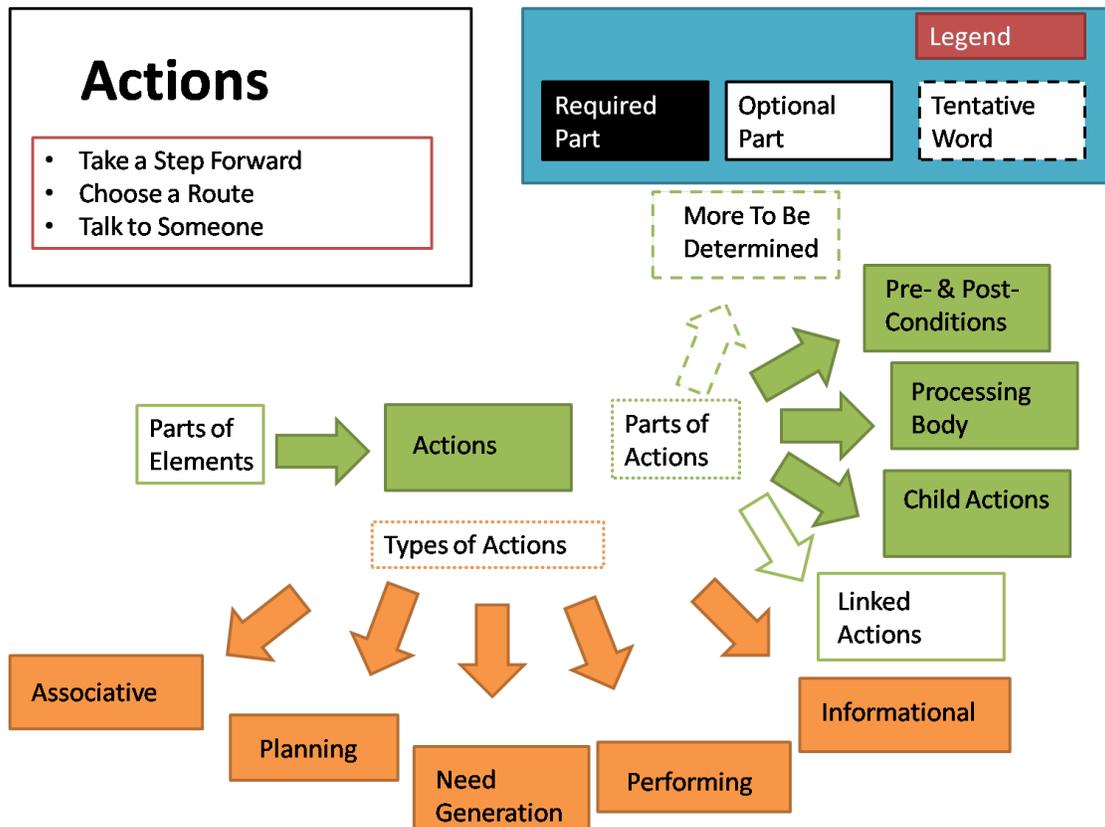
The language development process TRACC utilizes is an evolutionary one. A given category is targeted, spanning concepts within that category are listed, these concepts are then themselves organized with a minimum set of categorical words, based on this categorization process it is possible that the process will be refactored again from the top to take account of a more elegant organizational scheme. Therefore, the information in this section is likely to change as new insights emerge and the language evolves.



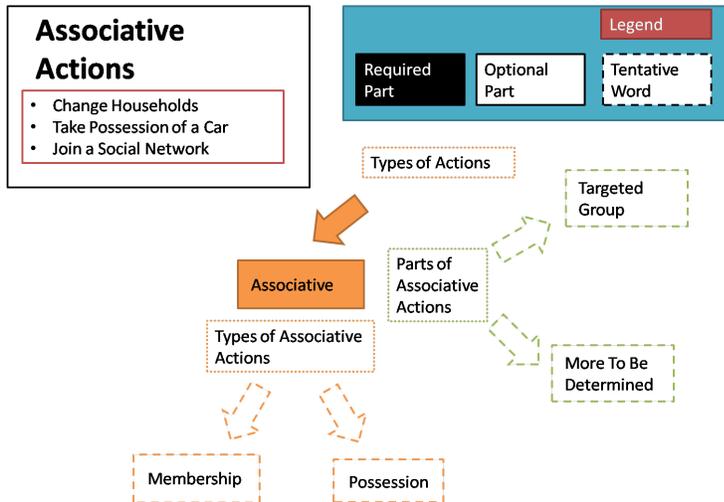
The element categorization has been explained above. Essentially, any “thing” in the transportation system can be dropped into one of three categories: Agent, Proto-Agent, or Environment. Every object in POLARIS which can be used in model development is an *Element*. *Elements* in POLARIS are any object which has any of:

- Attributes
- Actions
- Child Elements

The *Attributes* of an *Element* are various properties which describe its state, but which do not themselves have any of the above properties (i.e. an attribute cannot take any actions nor have any child elements, etc.). The *Actions* of an element are various functions which can alter either the internal state of the *Element*, or the external state of the system as a whole. Finally, *Child Elements* are lower-rank elements contained within an *Element*. The system of child/parent element relationships defines the element hierarchy of the object model, where all elements have parents, except for the root element – *WORLD*, and all elements have children, other than terminal elements which have none.

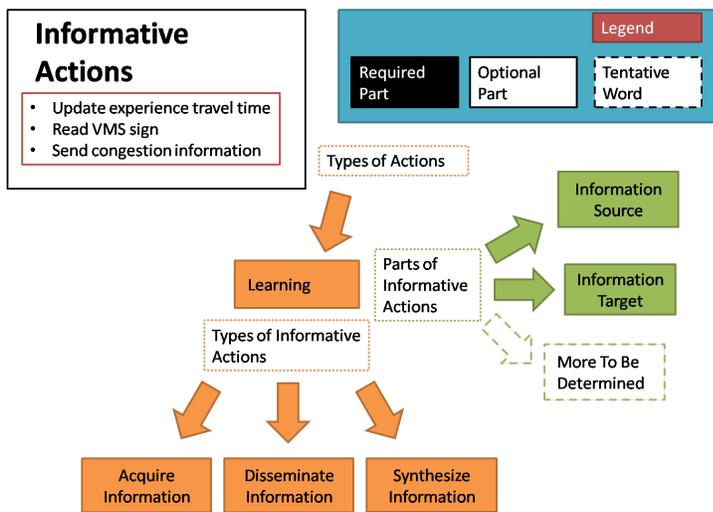


There are many actions which different elements can take, but they all fall under several common headings, and generally have either an internal (to the element) or external (system) scope. These five primary action types are:

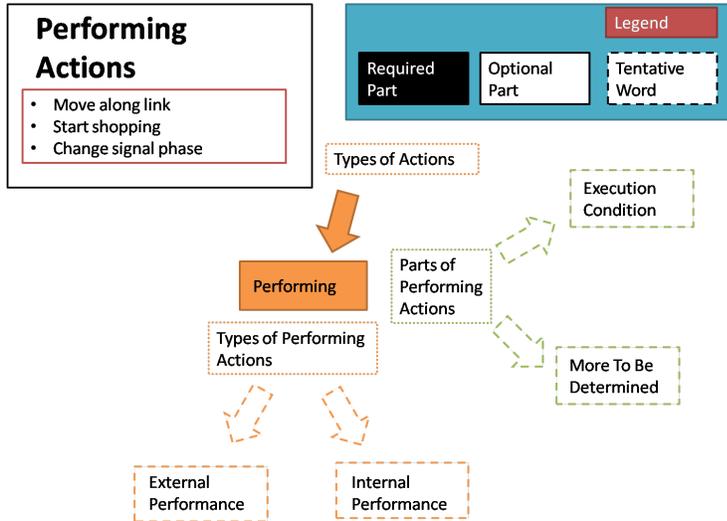


- Associative (Ownership/Membership/Joining/Leaving)
 - External Grouping Actions

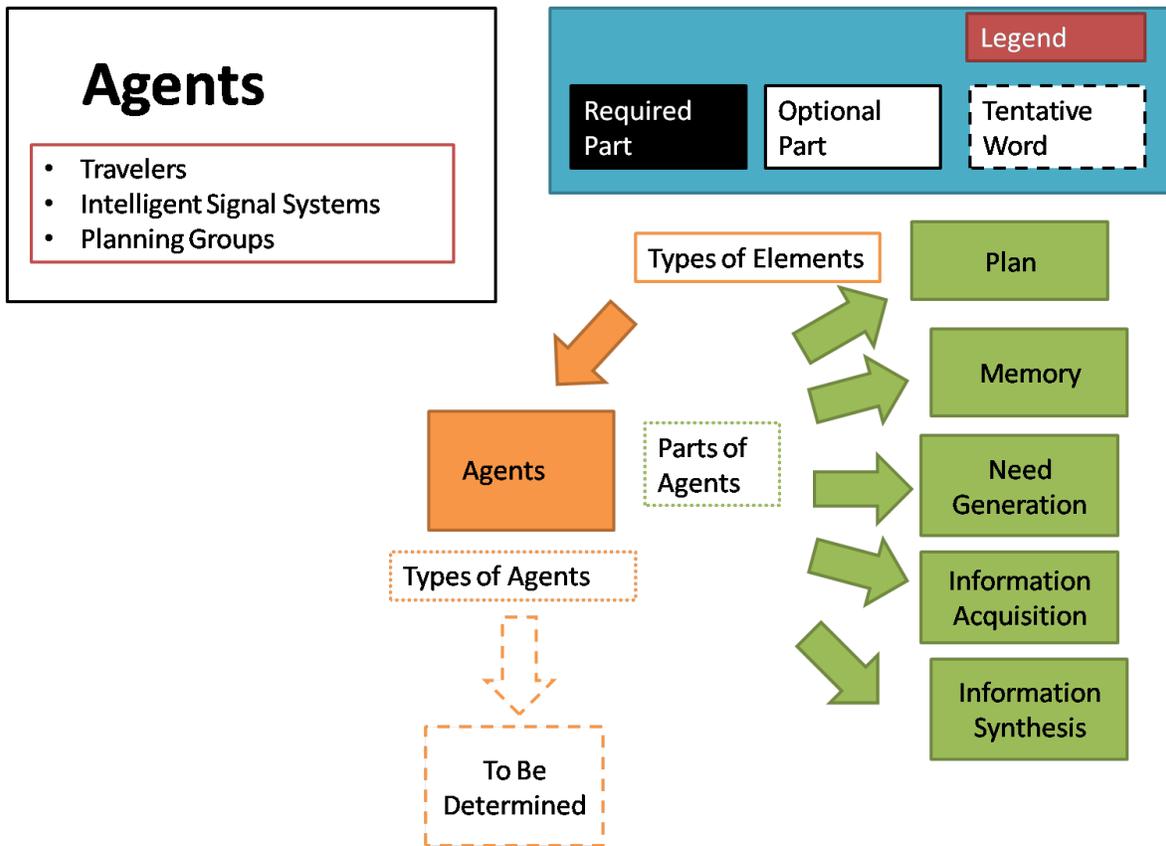
- Planning (change state of internal/external plans/formulate new plan attribute)
 - Planning External Actions
 - Planning Internal Actions



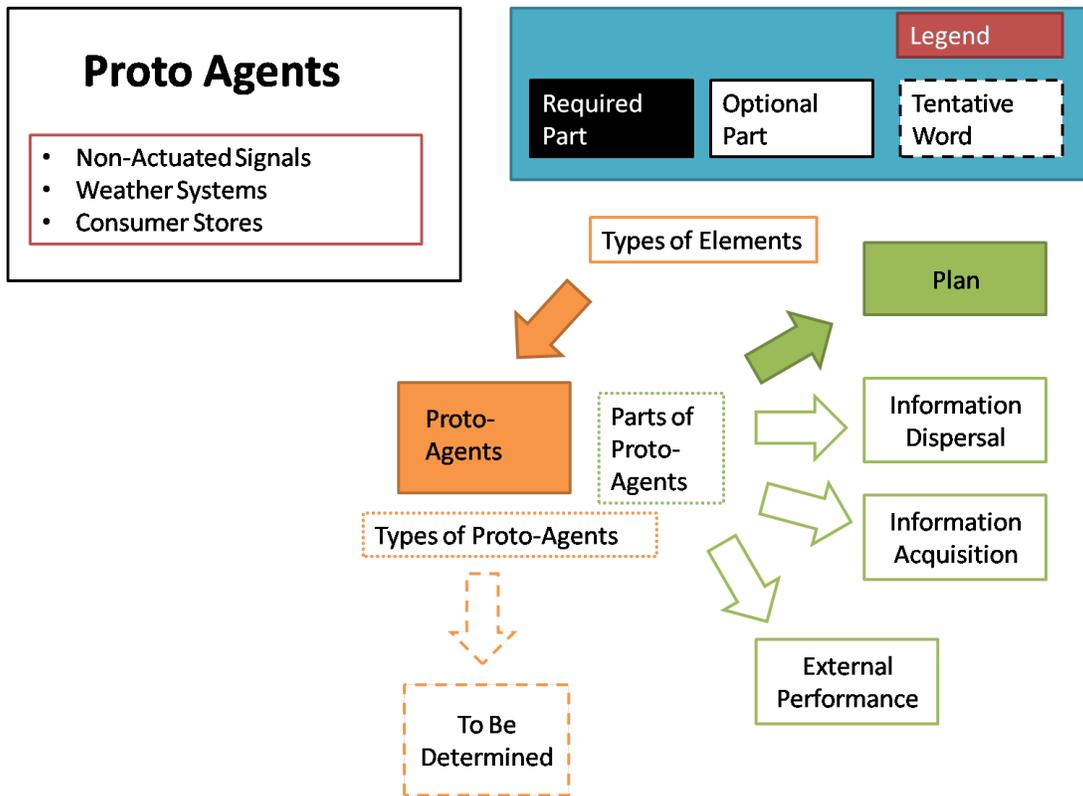
- Informational
 - Information Acquisition (Percept) Actions
 - Information Dispersal Actions
 - Information Synthesis



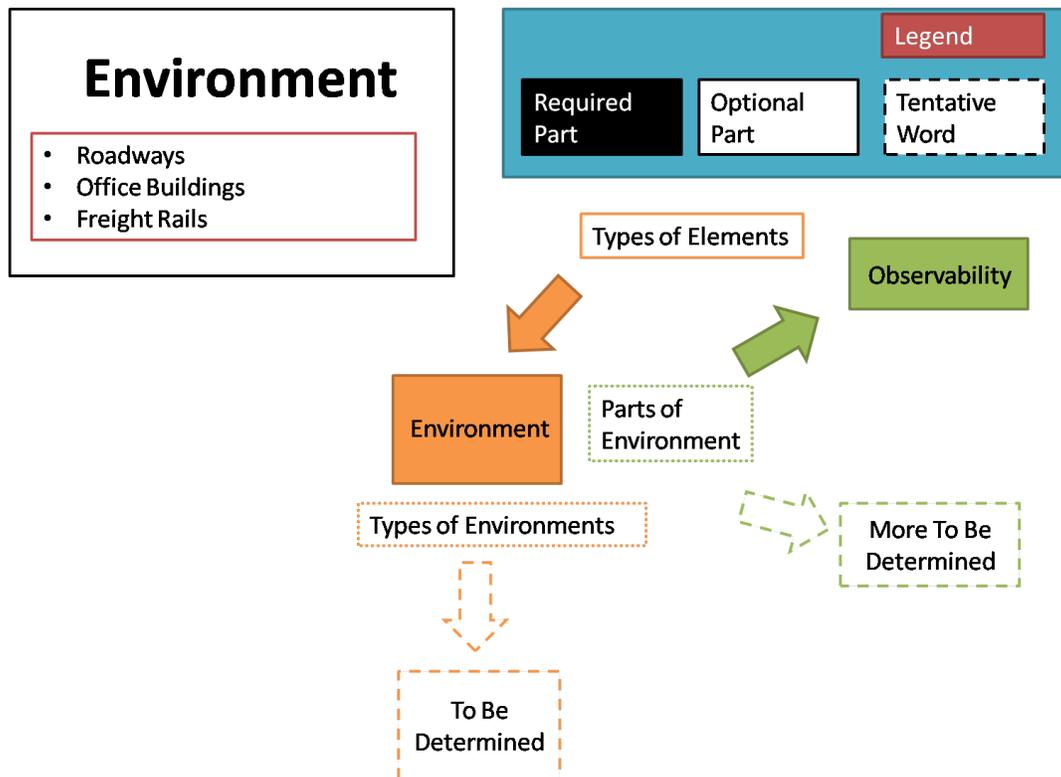
- Performing
 - Perform (Execute) external (to the element) Actions
 - Perform (Execute) internal (to the element) Actions
- Need Generation (Agent Specific)
 - Internal Need
 - External Need



Agents in POLARIS are any object which has at a minimum the Planning *Actions* and the Information Synthesis and Information Acquisition *Actions* defined. In other words an agent is something which can sense changes in environment and respond to it in some way. *Agents* also are required to have a *Plan*, which is a list of actions to be executed. This does not necessarily imply that agents are only living entities, as something like an actuated traffic signal is also an agent, in that it can sense information using the actuators, respond to changes in the system, and have plans put in place to do so. Agents also have a need or goal to satisfy, which can change over time. An optional attribute that many agents have is a *memory*, which stores individual agent impressions of the environment.



Proto-agents are similar to agents in that they can execute actions and have individual attributes, but differ in that no planning actions or information acquisition and synthesis actions are defined for a proto-agent. In other words, proto-agents do not respond to changes in the state of the system. Proto-agents do, however, have a *plan* to follow, the difference from an actual agent being that the proto-agent has no way to modify the plan.

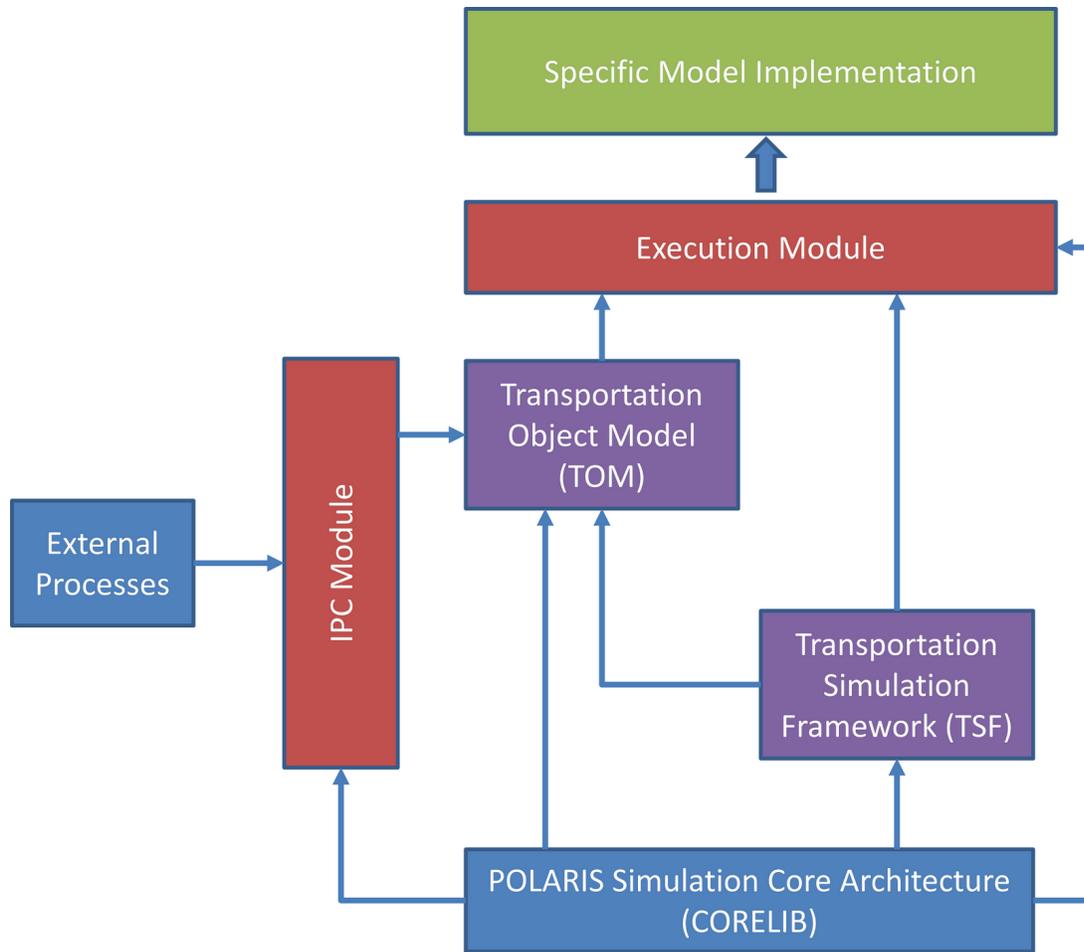


The final element type in POLARIS is the *Environment* element. Elements of this type are characterized by the lack of any defined actions, i.e. environment elements are inert in the system. Environment element attributes can change in the system, but not these changes are only made by external agents or proto-agents acting on the environment element.

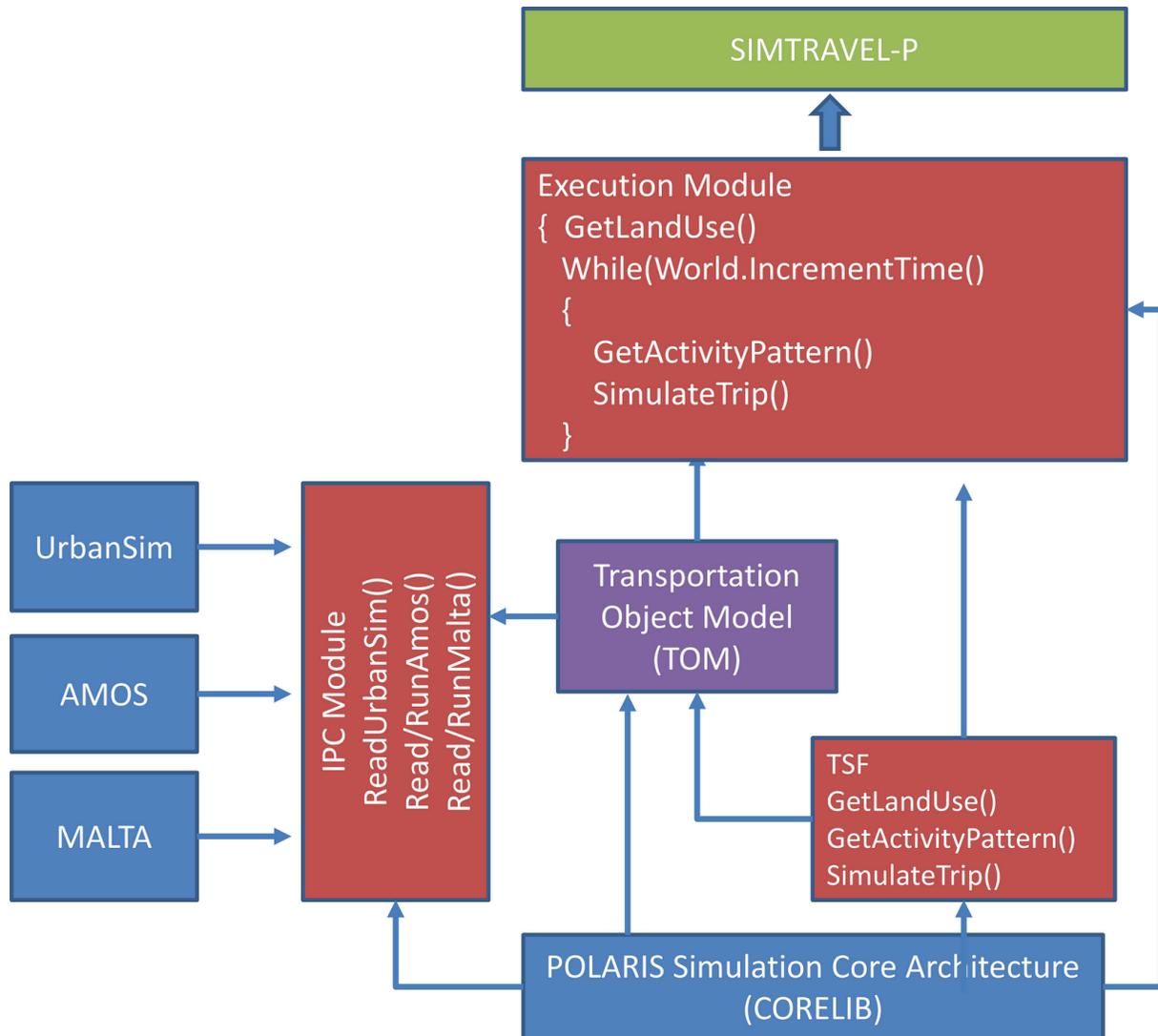
POLARIS Modeling System - Initial Design

The design of the POLARIS modeling language or model development system, is still in the early stages of conception, but borrows heavily from the agent-based modeling realm. A top level view of the core components that would comprise such a model development system is shown in Figure 1. The system is comprised of five primary components, including:

- Transportation Object Model (TOM)
- Transportation Simulation Framework (TSF)
- Interprocess Communication Module (IPC)
- Execution Module
- CoreLib



The main underlying concept here is that the Transportation Object Model describes the objects that would exist in a transportation simulation and their properties and actions, while the TSF would define those actions and CoreLib provides the low-level architecture for operationalizing the model. These modules would not necessarily be changed by users of the modeling language. The users have the option of reconfiguring the TOM and the TSF if they wanted to redefine any of the model actions, i.e. route choice, destination choice, lane change, etc. The IPC module, which determines how outside processes are called and how they share information, and the Execution Module, which lays out the flow of the simulation, would always need to be user-defined for any implementation. These user configurable modules are all written in terms of TOM objects and actions, and these are translated by CoreLib into actual execution code. Together the user implementations of these modules, or at a minimum the TSF and Execution modules, leads to a new implementation of a model in the POLARIS system. An example of a possible model implementation is shown in Figure 2.



This simplified example shows how the POLARIS architecture could be used to rebuild an existing model, the SIMTRAVEL model, in terms of the POLARIS language. In fact, the goal is such that any travel demand model could be replicated in POLARIS by rewriting minimal amounts of code. In the SIMTRAVEL example, the only code involved would be in converting the inputs/outputs of each external model into POLARIS objects, writing functions to call and get results from each model, and setting up an execution script to determine the order of model calls. This represents a simple implementation, where the processing is mainly handled external to the POLARIS system. Another option would be to rewrite AMOS, MALTA or both in the POLARIS language itself, which would be handled in the TSF and Execution modules, rather than as an external process. This would be the preferred method as it would be expected to give higher performance.

Transportation Object Model

The Transportation Object Model forms the basis of the POLARIS language from the user perspective.

The TOM is analogous to a dictionary for the POLARIS language. The Object Model describes all of the elements that are available to the modeler in POLARIS and defines them in terms of other elements and actions they can take.

For an initial listing of the “words” that will be contained in the TOM, consult the above section “POLARIS Core Categories”

The elements in the object model are related through their child-parent relationships and their external actions. The actions themselves are defined in the Transportation Simulation Framework (TSF).

Transportation Simulation Framework

The Transportation Simulation Framework (TSF) defines the operation of the various actions that the elements in the TOM can take in relation to each other. All of the various standard, and potential future activity-based modeling, routing, traffic microsimulation, land-use modeling, etc. concepts are defined here. Again, these concepts will all be defined in terms of the elements in the TOM acting on one another. The important concept in the design of the TSF is that it abstracts much of the low level detail of how the various simulation components are implemented away from the model developer. For instance, the model developer does not need to know where information about each agent is stored, how it is brought into and out of memory, how the simulation processes are dived into threads, etc. These implementation level details are all handled in the POLARIS CoreLib. The modeler only needs to know about the elements and actions defined in the POLARIS language. Only if new elements need to be added or redefined in a particular implementation are the lower level details in CoreLib modified. For implementing most transportation simulations, the goal is that modelers will have to go no deeper than the TSF.

Actions in the TSF are similar to elements in the TOM. Actions can have *Child Actions*, *Objects* and *Subjects*. The object of an action is the owner of the action, i.e. the element which controls the action, while the *subject* of the action is the element that the action is acting on. An action can also be reflexive, where the subject and the object are the same element.

A potential example of how the TSF would interact with the TOM and the Corelib can be seen in the action defined below.

In TOM: Agent type Person is defined with children elements schedule, which has child element activity

- World.Agents.Person
 - Attributes:
 - ...
 - Children: ...
 - Schedule
 - Activity
 - ...
 - Actions:
 - SetPlanTimes

In TSF: The actions of the Person agent are defined in terms of other elements.

```
Person.SetPlanTimes(Activity A, plan_type type)
{
    Attribute attr = this.Schedule.Activity(A).Attributes(mode_plan_time)
    If (this.IsEmployed && type==MODE) attr=World.Time
    Else attr = World.Time+1;
}
```

This set of TOM entries and TSF action definitions determines that there is a person agent in the simulation which determines takes an action to set a plan time for a later activity (this is a *Planning Action* since it only changes the state of the internal agent *Plan*). The action in the TSF is defined solely in terms of other elements, i.e. the schedule and activity child elements and the world parent element and the attributes of these. The CoreLib is then called during simulation to perform the required operations to carry out the TSF actions.

The TSF is likely to consist of standard methods covering the actions identified for the elements in the TOM, which can then be overridden in the TSF by individual model implementation developers. For example, if the route choice procedures defined for the standard agent do not conform to what a specific modeler needs, new versions of the route choice model can be implemented as long as the same inputs/outputs are used, so that the new route choice model fits seamlessly into the model flow. An example is shown below, where a different *GenerateActivity* function can be defined for an agent to produce either an AMOS-type or ADAPTS-type activity, respectively.

GenerateActivity() [in AMOS @ t]

```
{
    Activity A
    A.setPlanTimeStart(t+0)
    A.setPlanTimeStart(t+1/t)
    A.setPlanTimeMode(t+1/t)
    A.SetPlanTimeWhoWith(t+2/t)
}
```

- AMOS setup can use default generation of activities
- Uses small offset from current timestep to set processing order
- Activities planned as they are generated

GenerateActivity() [in ADAPTS @ t]

```
{
    Activity A
    A.setPlanTimeLocation(TSF.GetPlanTime(A,Loc))
    A.setPlanTimeStart(TSF.GetPlanTime(A,Start))
    A.setPlanTimeMode(TSF.GetPlanTime(A,Mode))
    A.setPlanTimeWhoWith (TSF.GetPlanTime(A,Who))
}
```

- ADAPTS overrides default activity generator
- Set planning times in simulation future
- Using new GetPlanTime function written in TSF

In the example, the AMOS-type generate activity function (this could also be TASHA, ALBATROSS, etc. with different attribute planning orders) says that when an activity is generated in the model its attributes are planned immediately along with it, with the location planned first, then the start time and mode are planned jointly, followed by the party composition. This type of activity planning is common to many existing activity-based models which do not rely on nested-logit type frameworks (which plan everything at once). In the second example, the attributes of the activity are not planned in the same

time as the activity is generated, but rather are scheduled as future events using the new `GetPlanTime` *action* which has been defined for the element, adding a level of planning dynamics to the activity model.

Inter-Process Communications Module

The IPC module is essentially the set of pipes which connect all disparate model parts together and allow them to talk to one another. IPC provides the critical capability of ensuring that all information transactions are well synchronized and thus inherently parallel. More importantly, the IPC provides the core competency of the framework to be interoperable by allowing communications between any of:

- Dynamic Files
- Static Files
- Shared Memory
- Distributed Memory
- Internet and Other Networked Systems

It can be thought of as an application of the Open Systems Interconnection model (a standard way of using layers to represent representing a communication system). As such, one of the initial models for generic exchange being considered for the IPC is based off of that used for internet connectivity - the following serves to illustrate the types of problems the IPC is concerned with:

- Initiator Communicator(s) Develops Communication Need and Begins Process
- Locate Communication Target(s)
- Read Target(s) Communication Protocol to Determine How to Engage
- Attempt to Engage Target
- Initiator and Target Develop a Communication Strategy
- Act to Develop a Secure and Stable Channel
- Package Information
- Send/Receive Information through Channel
- Verify Completion
- Terminates Communication

In this case one can see that the language is general enough to apply equally well whether the communication is between two agents next to each other on a road or an activity planning routine examining a file being dynamically written by another microsimulator.

Execution Module

After determining the elements of the simulation model and the actions they can take in the TOM and TSF modules, or simply using the default TOM and TSF, the modeler will then build the specific model implementation in the Execution Module. This module determines how the simulation is initialized, how it proceeds, what the outputs of the simulation are, terminal conditions, etc. This section, then,

represents the formal description of the actual transportation simulation. Again, all elements of the execution module are written in terms of TOM elements and actions operating on TOM elements from the TSF or the IPC. The execution module runs according to the model script developed by the user and updates the state of the modeled system. The combination of a set of external processes, elements, internal actions and a model script in the execution module, define a fully implemented simulation model.

CoreLib

The POLARIS CoreLib serves multiple functions in the framework. Its' primary job is to be able to translate the high level words of the Transportation Object Model and Transportation Simulation Framework into efficient C++ operations. Another function is to provide efficient data containers and sub-systems which are specially suited to the needs of transportation and agent-based problems (such as hash maps which can be pre-sized for re-use or a serialization library which can be applied to seamlessly warehouse agent learning). It is also tasked with providing a set of diagnostic and code organization APIs (such as "read-only" variables or performance timers). The CoreLib will also provide common utility functions such as time conversions, unit conversions, re-projection, and so forth. Common file formats will be stored and processed by CoreLib.

Collaboration

Meeting with Scientists in Decision and Information Sciences Division

TRACC hold a two-day meeting with scientists in Decision and Information Sciences (DIS) Division at Argonne on agent-based modeling and simulation. DIS scientists include Dr. Charles Macal, Dr. Michael North, Dr. John Murphy, Eric Tatara, and Jonathan Ozik. Dr. Charles Macal is the director of the Complex Adaptive Systems Group in DIS. Dr. Macal is an internationally recognized leader in the field of agent-based modeling (ABM) for his development of novel large-scale ABM applications and the development and support of the open access agent-based modeling toolkits. Dr. North is the lead developer of Repast, the Recursive Porous Agent Simulation Toolkit (available at <http://repast.sourceforge.net/>), was been developed and is supported by Macal's CAS2 group at Argonne National Laboratory. Repast is a leading open-source agent-based modeling toolkit, available at no cost, that is being used in numerous application domains, including the modeling of health care systems and disease transmission. Research on Repast by Macal's group includes the development of a version of Repast for high-performance computers (Macal et al., 2008).

Traffic Simulation Seminar

TRACC invited Dr. Samer Hamdar, an assistant professor and the director of traffic and networks research laboratory at George Washington University, to present his work on traffic simulation models. He is an affiliate faculty at the Center of Intelligent System Research (CISR) and the National Crash Analysis Center (NCAC). He holds a M.S. Degree from the University of Maryland, College Park and a Ph.D. Degree from Northwestern University – both in Civil and Environmental Engineering - Transportation. Dr. Hamdar worked on different projects covering different transportation areas. These projects include two National Science Foundation (NSF) Projects titled "Modeling Driver Behavior from a Cognitive Perspective" and "New Methods for Measuring, Evaluating and Predicting the Safety Impact of Road Infrastructure Systems on Driver Behavior"; and a Federal Highway (FHWA) Project

titled “Incorporating Weather Effect in Traffic Estimation and Prediction”. His primary research interests include Driver Behavior Modeling, Traffic Flow Theory, Intelligent Transportation Systems, Transportation Planning and Evaluation, Transportation Safety, Evacuation Modeling and Disaster Management. He has an international research background having participated in projects in Germany, Saudi Arabia and the USA.

Dr. Hamdar’s presentation topic is *From Cognitive-Based Decision Making to Car Following Modeling: Beyond an Accident-Free Environment*. Most existing traffic simulation models assume travelers with homogeneous behavior and ignoring the stochastic fact in real world. Dr. Hamdar’s research relaxes this assumption and allows user heterogeneity in microscopic traffic simulation models to allow stochastic simulation of driving behavior, e.g. car following and lane changing.

The abstract of Dr. Hamdar’s seminar is as follows.

In the year 2005, the monetary cost of injuries related to traffic accidents reached 625 billion USD (U. S. Dollars) in the U.S.A. only. Based on the National Highway Traffic Safety Agency (NHTSA) studies, 5 accident types of interest can be identified: 1) rear impacts (29.6% of US accidents), 2) angle or side impacts (28.6 % of US accidents), 3) fixed object crashes (16.1 % of US accidents), rollovers (2.3% of US accidents), head-on collisions (2 % of US accidents) and collision with pedestrians/bicyclists (1. 8 % of US accidents). In car-following, the focus is on the tailgating behavior that may lead to rear-end collisions. However, existing car-following models are built in an accident-free environment with a structure unsuited to capture driver behavior during incident scenarios; relaxing the safety constraints in these models causes breakdowns with an unrealistic number of chain-type accidents.

In 2008, an acceleration model was formulated by Hamdar et al., so that it incorporates the risk-taking attitudes of drivers; while using prospect theory to evaluate the perceived consequences of applying different acceleration rates, a probability of collision and a crash penalty term are explicitly introduced in the formulation. The objective is to explore the characteristics of this formulated car-following model in terms of its ability to capture congestion dynamics and the collective accident-prone behaviors on a freeway section. After being calibrated against real-life trajectory data, different incident scenarios are modeled including rear-end collision and fixed object crashes (Type 1 and Type 3 above). The effect of both psychological factors and execution/perception errors on the accidents number and their distribution along a freeway length is studied. Through sensitivity analysis, correlations between the crash-penalty, the negative coefficient associated with losses in speed, the positive coefficient associated with gains in speed, the driver's uncertainty, the anticipation time and the reaction time are retrieved. The formulated model offers a better understanding of driver's behavior under extreme/ incident conditions with a better insight into the psychological/cognitive reasoning adopted. Accidents are created as an inherent result of the utility function without imposing unnecessary safety constraints to prevent them.

Next Steps

- Complete literature review of existing transportation models
- Complete listing of elements and concepts to include in Transportation Object Model
- Develop Transportation Object Model language elements, including element hierarchy, definitions, attributes, etc.

- Complete listing of existing and near term transportation processes to include in framework
- Formulate process in terms of POLARIS language elements
- Begin development of POLARIS language specification