Activity Theory Language as a Visual Studio 2013 Tool Extension

Geri Georg
Computer Science Department, Colorado State University, Fort Collins, Colorado
`georg@cs.colostate.edu,`

November, 2014

Technical Report CS-14-102

Activity Theory Language as a Visual Studio 2013 Tool Extension[1]

Geri Georg[2]
November, 2014

1. Introduction

This technical report details an Activity Theory (AT) language domain specific language (DSL) tool implemented in Visual Studio (VS) using the DSL Modeling Software Development Kit (SDK). Changes and extensions to the initial AT language that were needed for this project are presented. We also discuss extensions that were made to the language as a result of using the tool to explore three example systems. We plan to use this tool to study additional characteristics of AT including contradiction analysis and mapping to Goal models, both of which were the subject of experiments in this tool, and are included in it to a limited degree.

In previous work we defined a Unified Modeling Language (UML) metamodel for an Activity Theory (AT) language, and also realized it as a model in the UML-based Specification Environment (USE) tool [2-4]. We additionally proposed tracelink mappings [8] from AT model elements to Goal-oriented Requirement Language (GRL) model elements [5], proposing that AT and GRL exhibit synergistic characteristics that can be exploited as part of the requirements engineering process to make more explicit the social constraints and requirements of complex systems that involve both human and computing components. The systems of interest to us are those with a wide variety of human stakeholders with disparate backgrounds and goals, especially systems where data is gathered directly or indirectly from some stakeholders and is used to manipulate or influence their environment. In these cases the groups of stakeholders are often not aligned towards the same goals, and the resulting systems therefore may not address critical requirements of some group. The success of these types of systems is often dependent on how well they address the different social requirements of their stakeholders. Our hypothesis is that using a psychological framework such as AT may help reveal the social requirements of these types of systems and ultimately positively influence their success, both during initial development and later during system evolution. A critical component of testing this hypothesis is the ability to automate the use of AT to produce repeatable results for a variety of systems, especially for persons not familiar with the AT framework. Thus, the need for automated tools is crucial to testing our hypothesis.

The USE realization of the AT language was useful in initially developing the language and its metamodel. However, there were several limitations of this realization. First, a user must be comfortable with the Object Constraint Language (OCL) and interpreting issues with an object model and any OCL constraints that it might violate. Second, creating an object model is quite detailed. Third, the visualization of an object model can be complex and difficult to interpret. Fourth, AT depends on natural language a great deal in order to help an analyst find issues in an AT model, most specifically in conflicts that are evident in the modeled activity, and natural language is not analyzed easily using any formal constraint language such as OCL.

[1] This report uses color in diagrams on the following pages: 25, 28, 29, 31, and 36.

[2] Computer Science Department, Colorado State University, Fort Collins, Colorado

This technical report details a second realization of the AT language using Visual Studio 2013, and its associated Domain Specific Language Modeling SDK. The result of this work is a VS extension for AT. The motivation for this realization using VS to see if a VS implementation can: 1) improve the object model creation complexity as experienced with the USE tool, 2) make use of the DSL development framework available in VS which generates a graphical editor for the language along with providing validations based on the language metamodel and other user-provided rules, 3) gain experience developing a graphic editor for a DSL, 4) explore analysis of various relationships in an AT model above and beyond structural constraints using custom C# code extensions to the generated code, and 5) explore the mapping of AT model elements to GRL Goal models and identify the extent to which Goal model elements and their relations can be deduced, based on the elements and relations present in the AT model.

To use any AT tool, a requirements engineer or system analyst will create Activity Theory System Diagrams (ASDs) from stakeholder input. The basic form of an ASD is taken from Engeström's work [1]. (Please see Appendix A for some background on Activity Theory.) In the VS AT tool, the ASD elements and their relations, as defined in the language metamodel, can be used to determine whether an ASD is well-formed, through structural validation and additional coherence and validation rules. Missing relations can be used to decide where e.g. further input is needed from stakeholders to resolve ambiguity or contradictory information. Once ASDs are defined, trace-link mappings can be used to transform them into a URN Goal model (using the GRL notation) in the jUCMNav tool [6]. Similar to the USE AT realization, the VS AT realization cannot create the GRL elements directly, but we have added report generation capabilities to the VS AT tool that specifies GRL elements and relationships based on the AT model. (Please see Appendix B for more information on trace-link mappings.) Goal and trade-off analyses are available in jUCMNav and these can be used to further refine the Goal model. jUCMNav supports powerful tracing capabilities from Goal model elements to high-level designs specified using URN use case maps, or UCMs.

Multiple examples are used in this document: (1) a case study system we explored while creating initial versions of the AT language, (2) a simple vacation activity that illustrates several of the AT language concepts, and (3) an electronic task booklet proposed to replace paper task booklets in a professional degree program at our university. The last example prompted additional relations to be added to the AT language metamodel as a result of exploring GRL mapping. Each of these examples is discussed in the report.

This report is structured as follows. Section 2 contains a description of some of the features of the DSL SDK. Section 3 briefly discusses major the tool design guideline. The general architecture of the tool is presented in Section 4, including the definition of the AT language as a metamodel, along with its specification in VS. The structure of the tool realization using the DSL framework, along with user commands that were developed during the project are also presented in this section. Some particulars of the implementation are discussed in section 5, then the examples used during development of this tool are presented along with tool functionality they illustrate or influenced in Section 6. Section 7 discusses some issues and limitations of this work and how some of them might be addressed in the future, and our conclusions are presented in Section 8.

2. Developing a Visual Studio Domain Specific Language

The DSL modeling SDK provides a graphical editor to specify the metamodel of a DSL, along with the graphical editor shapes that will be used by end users to specify instance objects of the DSL. The metamodel and shape specifications are separately defined, and a mapping relation is used to associate them. Both language domain classes and relations may have associated shapes that will be accessible through the language graphical editor. Different types of shapes are available to represent language domain classes; geometry and compartment shapes are used for the AT language. Compartment shapes allow sets to be displayed, for example the names of all the Tools related to a specific ASD. Both geometric and compartment shapes can contain icon and text decorators, which may be displayed based on the value of some attribute of the language domain class. For example, if a mediation is defined between a Tool, Subject, and Aim, the mediation type can be set to an enumeration literal associated with Subject/Aim and a particular icon displayed in the mediation shape.

The language metamodel must be specified along with associated diagram shapes and the mapping between the language concepts and these shapes. Beyond these items, enumerated types may be defined for the language. Graphical tools must also be added to the DSL language graphical editor toolbox to create different domain class objects or relation instances.

There is extensive support for customization using C# to augment the default behavior of generated code. C# allows partial classes to be defined so that the custom code can be separated from the generated code. Custom code is used in the VS AT tool to add constraints that are not structurally possible in the metamodel, for example the constraint that at least one of each type of ASD element is required for a well-formed ASD. Another constraint is that each mediation needs to be related to two mediated ASD elements. (The multiplicities allowed in the metamodel are 0..*, 0..1, 1..1, and 1..*, so a multiplicity of 2 must be specified as either 0..* or 1..* in the metamodel, then custom code must be added to the relation builders to enforce the multiplicity restriction.)

The DSL SDK supports validation, and this is used in the AT tool on model open, save, or when selected from the user command menu. The default validation is based on the metamodel structure, but again custom code can be added to augment this behavior. Custom code has been added to check that at least one of each type of ASD element is related to an ASD.

Custom code can also be added to enforce Rules, which are defined to fire when various model elements or relations are created or deleted. An example of a rule is one used to change the enumerated type of a mediation. The fact that a mediation is related to a Subject and Aim can only be known when the second relation from the mediation to a mediated element is set, and at this point the mediation enumerated type attribute is set to 'SA' which allows a related icon to be displayed in the mediation shape.

Custom code can be used to manipulate model elements in the persistent store in addition to the display elements that are associated with them. This ability is used extensively to realize the user commands for the AT language tool, and custom code is also used to add classes such as dialog box classes that are used by commands to read from or write to a file.

All of these features are discussed with more detailed examples in the following sections.

3. Major Design Guideline

This realization of the AT language was developed as a research aid, and as such has weaker requirements than would be present in a tool meant for general use. In particular the full set of well-formed constraints needed to support some automatic analyses were relaxed so that models could be developed and manipulated incrementally. Therefore, for all but the most fundamental structural constraints (e.g. an ASD must have at least one of each type of ASD element associated with it), multiplicity constraints and custom code to validate them have been weakened so that they do not introduce validation errors. Instead, user commands that analyze the stronger constraints have been added and implemented through C# methods and various reports to alert the modeler of potential issues which can be addressed or ignored as the modeler chooses. The rationale is:

(a) while complete models are needed for deep analyses, a researcher or modeler may not want to spend much time initially adding detail that may be subject to frequent change.

(b) partial model entry, where only a few constraints are enforced on model open and save, may be more user-friendly.

Specifically, the relation multiplicities that are weakened in the VS AT language metamodel are shown in Table 1. The first column gives the name of the relation as identified in the Visio diagram of the AT language metamodel (shown in Figure 1). The related multiplicities for the classes in the relation are shown in the next two columns. The last two columns show the multiplicities for the relation between these classes in the Visual Studio DSL metamodel (shown in Figure 2).

Table 1. Relaxed multiplicity relations in Visual Studio AT language metamodel

Relation name in Figure 1 metamodel	**Fig 1 source multiplicity**	**Fig 1 target multiplicity**	**VS metamamodel source multiplicity**	**VS metamodel target multiplicity**
eleInASD	1..* ASD	0..* ASDElement	0..* ASD	0..* ASDElement
tools2dols	0..* Tool	1..* DoL	0..* Tool	0..* DoL
rules2dols	0..* Tool	1..* DoL	0..* Tool	0..* DoL
whoDoesDoL	0..* DoL	1..* Community	0..* DoL	0..* Community
subjComm	0..* Subject	1 Community	0..* Subject	0..1 Community
mediationRel	0..* Mediation	1 MediaTINGEle	0..* Mediation	0..1 MediaTINGEle
mediatedRel	0..* Mediation	2 MediaTEDEle	0..* Mediation	0..* MediaTEDEle
medRelASD	1 ASD	1..* Mediation	0..1 ASD	0..* Mediation

4. AT Tool Development –AT Language Metamodel

A UML diagram of the AT language metamodel is shown as a Visio drawing in Figure 1. This figure has three distinct parts. First, the DSL Modeling SDK requires a model root, so a class was added that serves as the root of a model (ATModelRoot). Every element in the model must be related to only one model root.

Second, there are three classes, DiagColorKey, AnalysisError, and Contradiction which are not a part of the AT language, but serve to hold additional information about a model. The purposes of these classes are as follows.

Currently two DiagColorKey objects may be defined for a model. One holds text explaining the colors used for various relations in an AT model, (for example, relations from an ASD to a related ASDElement are colored in Gainsboro). The second object holds text explaining the color used for different model analysis problems, (for example, any mediating element that is not involved in a mediation is outlined in Red).

An AnalysisError object is created whenever an analysis is run via a user command, and a problem is found, for example, that a mediating element is not involved in any mediation. Such a situation may indicate either that the mediating element should not be related to the ASD or that a mediating relationship should exist that has not yet been specified, or that some mediated element that would be part of such a relationship has not been added to the ASD. In general, it is a decision on the part of the modeler as to which of these possibilities is true, and how it should be addressed or whether it should be ignored. There are several types of problems that can be present, and so each AnalysisError object has an enumerated type attribute that indicates the error type.

A Contradiction object is user-defined at this time, and can be useful to identify potential problems in a model. AT contradictions can arise between ASD elements of the same type, across different types, between ASD outcomes in one ASD and some other ASD element type in another, and between evolutions of a single ASD. We experimented with the Apache Natural Language Processing (NLP) tools to help identify contradictions in the natural language description of ASDElements, but did not arrive at a satisfactory solution. Therefore the decision was made to proceed with an entirely user-defined contradiction object at this time (the NLP experiment is describe in the Issues section of this report). A contradiction is always relative to some item, either an ASD or some ASDElement. This is the considered item related to a contradiction. There can be any number of other ASDs or ASDElements that contradict this considered item.

The third part of the Figure 1 consists of the AT language classes. These are: ASD, ASDElement, and Mediation. ASDElement is an abstract class with three specializations: MediaTINGEle, MediaTEDEle, and Outcome. MediaTINGEle and MediaTEDEle are also abstract classes. Tool, Rule, and DivisionOfLabor elements are specializations of MediatinTINGEle, while Subject, Aim, and Community elements are specializations of MediaTEDEle. A Mediation object must be related to one mediating object, and two different mediated objects, all of which must be related to the same ASD as the Mediation object. The Outcome class is only a specialization of ASDElement, and an Outcome object is considered neither a mediating or mediated element of an ASD.

Additional constraints in Figure 1 are written in English text with arrows pointing to the relation or class which is constrained. (In the USE realization of the AT language, these constraints are implemented as OCL constraints on the AT USE model. In the VS tool they must be implemented as C# custom code added either to validation or relation builder methods, as described later in this paper.)

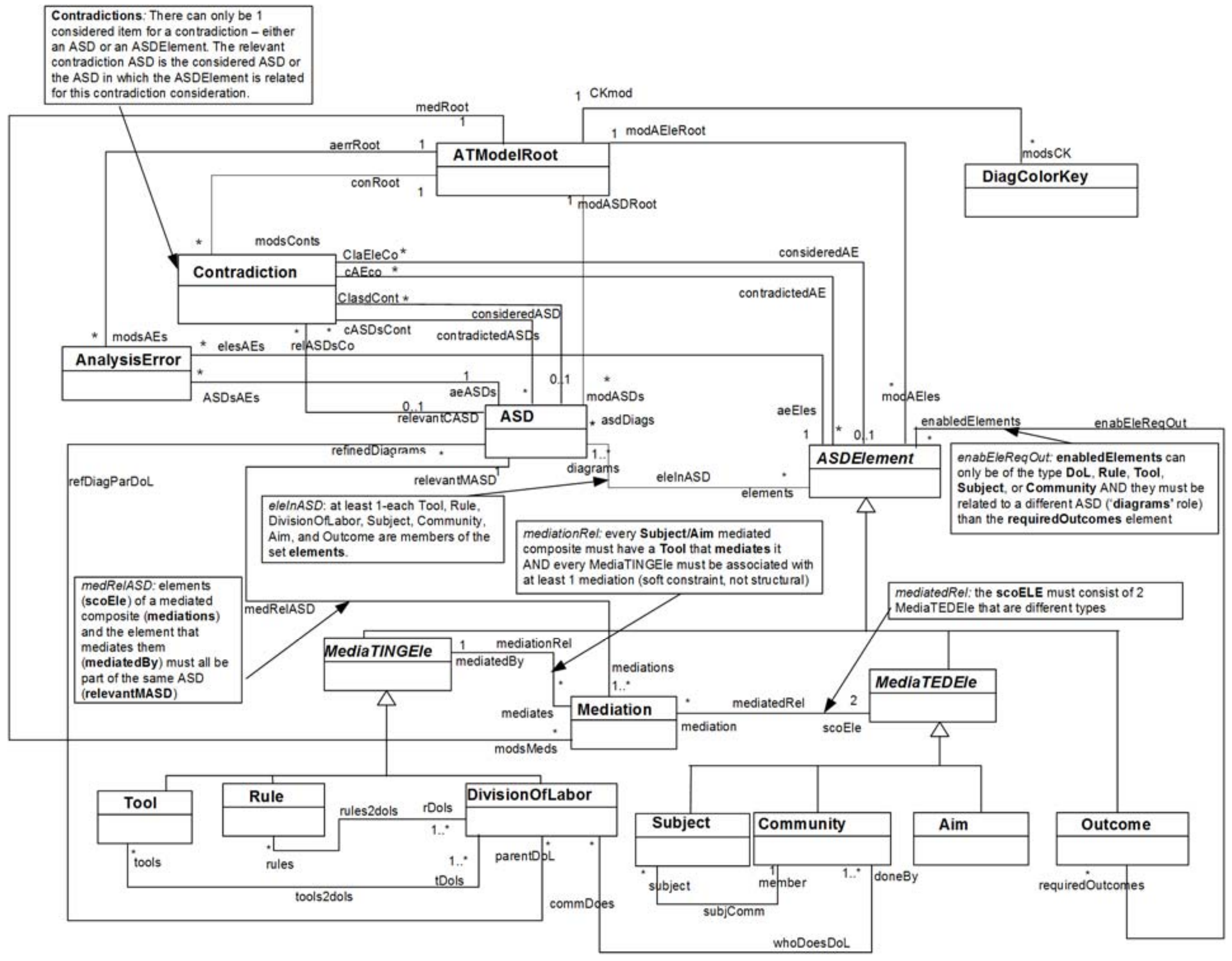

Contradictions: There can only be 1 considered item for a contradiction – either an ASD or an ASDElement. The relevant contradiction ASD is the considered ASD or the ASD in which the ASDElement is related for this contradiction consideration.
ATModelRoot
DiagColorKey
Contradiction
AnalysisError
ASD
ASDElement
MediaTINGEle
Mediation
MediaTEDEle
Tool
Rule
DivisionOfLabor
Subject
Community
Aim
Outcome
eleInASD: at least 1-each Tool, Rule, DivisionOfLabor, Subject, Community, Aim, and Outcome are members of the set elements.
mediationRel: every Subject/Aim mediated composite must have a Tool that mediates it AND every MediaTINGEle must be associated with at least 1 mediation (soft constraint, not structural)
enabEleReqOut: enabledElements can only be of the type DoL, Rule, Tool, Subject, or Community AND they must be related to a different ASD ('diagrams' role) than the requiredOutcomes element
medRelASD: elements (scoEle) of a mediated composite (mediations) and the element that mediates them (mediatedBy) must all be part of the same ASD (relevantMASD)
mediatedRel: the scoELE must consist of 2 MediaTEDEle that are different types

Figure 1. AT Metamodel

Visual Studio AT DSL Metamodel

The first step in developing a VS DSL is to diagrammatically develop the metamodel, which consists of some root class, additional domain classes, and relationships. The root of the DSL must have embedding relationships with all other classes in the DSL. The ATModelRoot domain class is the root of the DSL metamodel. This metamodel specifies abstract domain classes for ASDElement, MediaTINGEle, and MediaTEDEle, similar to the metamodel of Figure 1.

The second step in developing a VS DSL is to diagrammatically specify the graphical editor shapes that will be used to visualize a model created using the DSL. Any model element that needs to be included in a visualization must have a shape defined and then mapped to it. Multiple domain classes can be mapped to a single shape, e.g. an ASDElementShape is defined and each of the concrete ASDElement specializations are mapped to it. Each element has an attribute that holds its concrete type as an enumeration, e.g. TOOL for a Tool element, and a related identifying icon is visible in the shape depending on this enumeration literal.

The AT DSL metamodel is shown on the left side of Figure 2, and diagram elements defined for the language are shown on the right side of this figure. Lines from elements on the left to elements on the right indicate mapping from domain language items to display shapes.

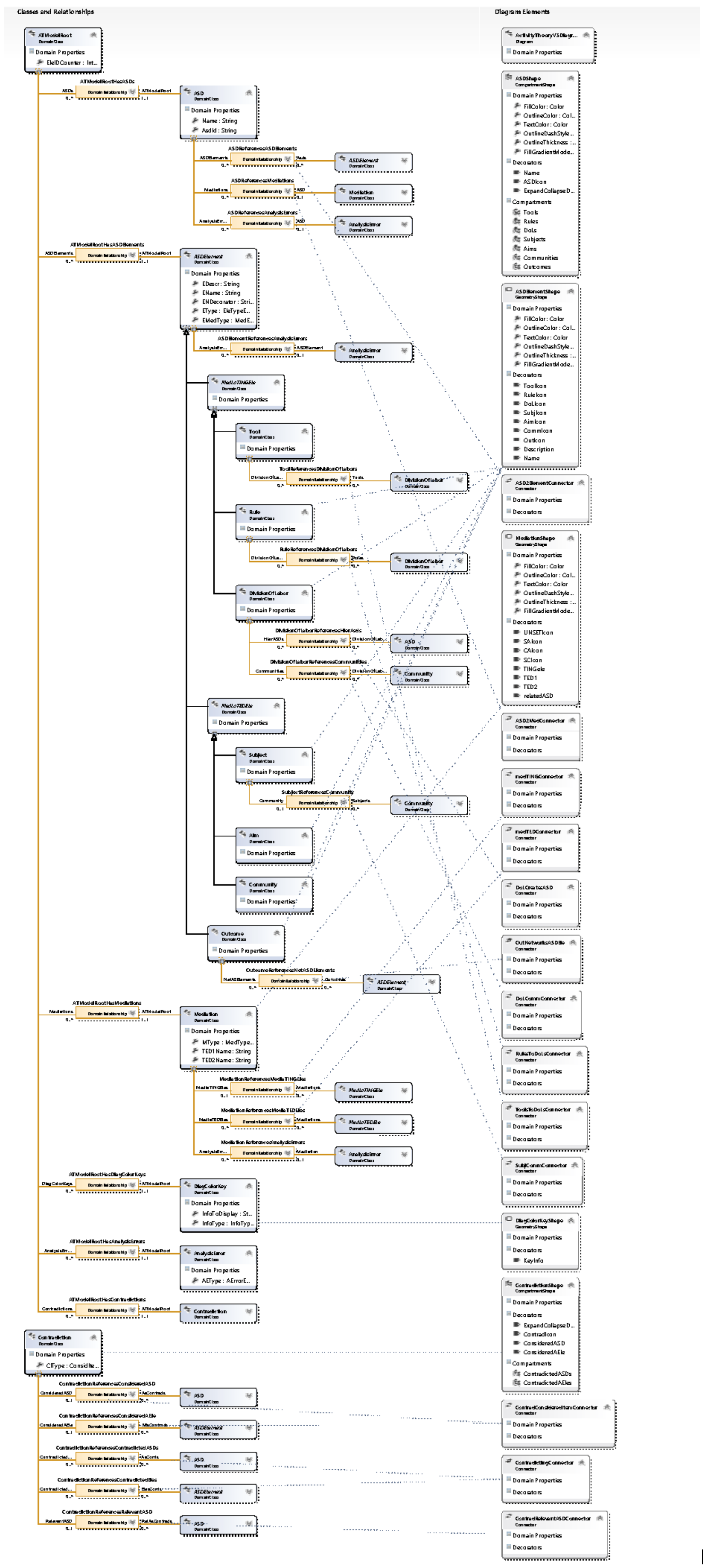
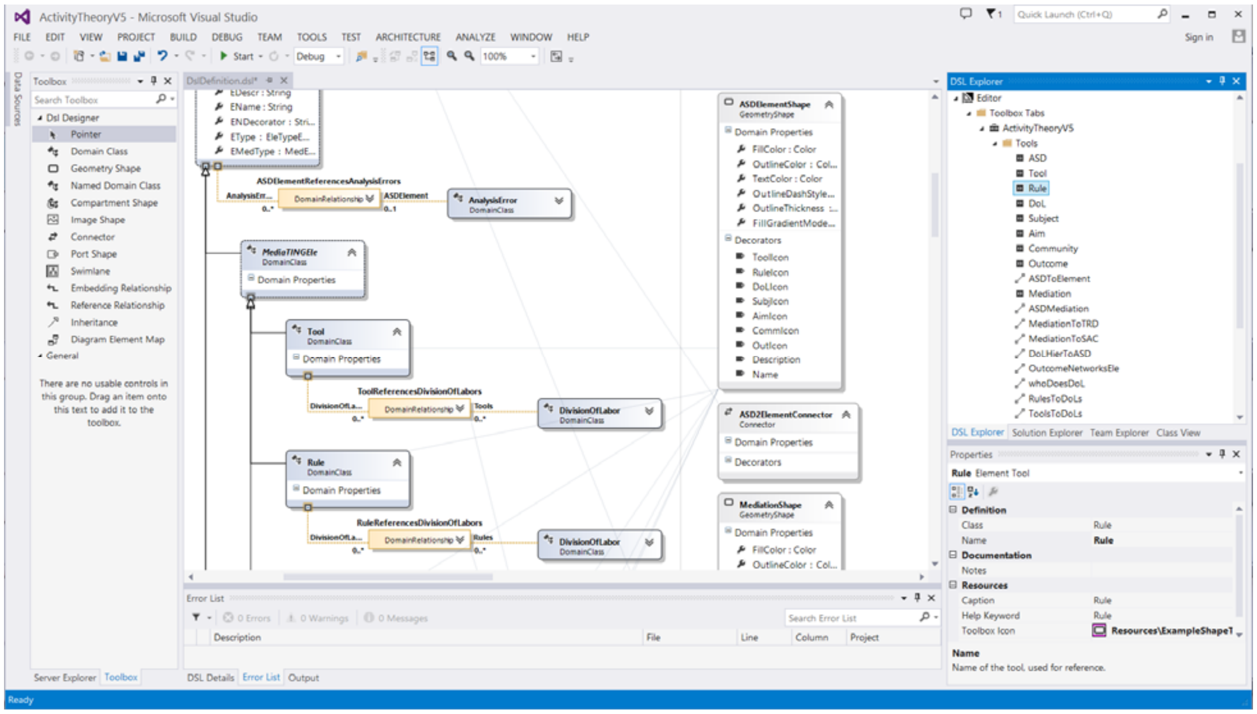


Figure 2. VS AT Metamodel

The next step in developing a VS DSL is to define graphical editor tools that an end user of the DSL will be able to use to create models in the DSL. Two types of tools are available, a connection tool and an element tool. Figure 3 shows this information specifically for a Rule ASDElement.



Properties

Rule Element Tool

Definition

Class	Rule
Name	**Rule**

Documentation

Notes	

Resources

Caption	Rule
Help Keyword	Rule
Toolbox Icon	**Resources\ExampleShapeT**

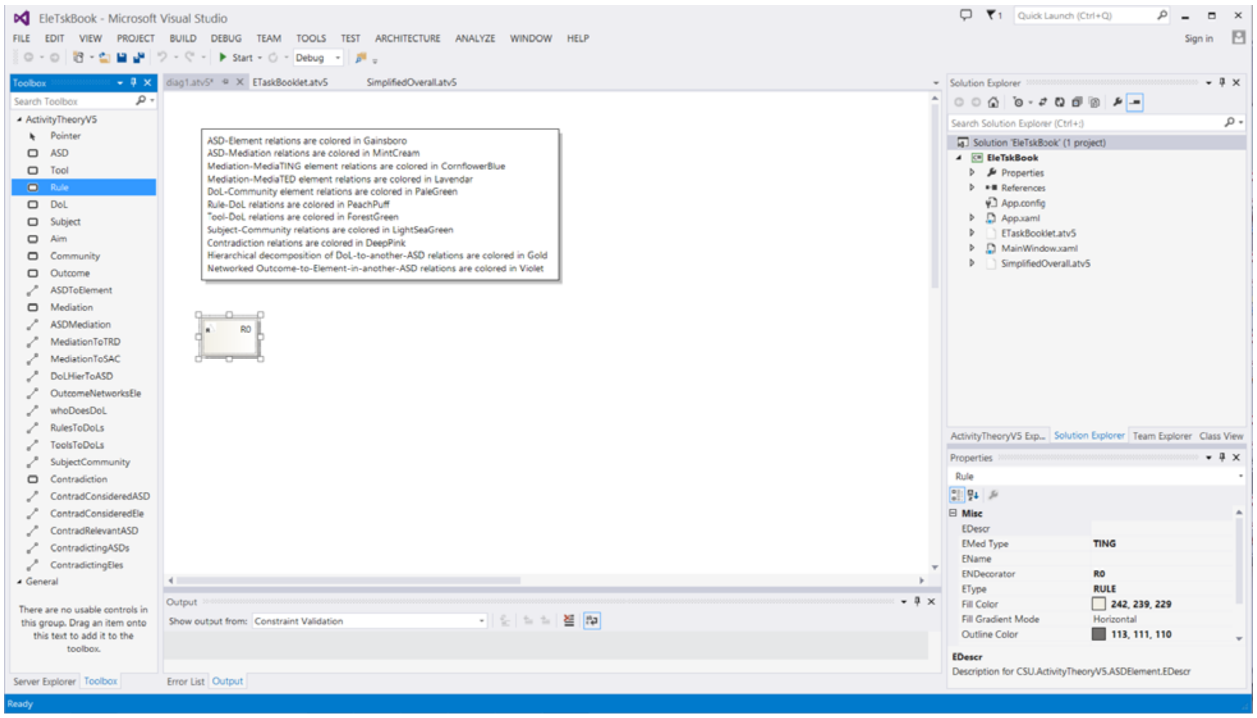
Name

Name of the tool, used for reference.

Figure 3. Screenshot of Rule portion of VS DSL metamodel and Toolbox definition for Rule tool

A portion of the VS DSL metamodel showing the Rule concrete language domain class is in the left part of the main window of Figure 3. Its mapped shape, ASDElementShape is shown on the right side of this window. The DSL Explorer tab is shown to the right of the main window, with the Editor->Toolbox Tabs->ActivityTheoryV5->Tools->Rule selected. The properties for this tool are shown in the lower right Properties tab. This shows that a Rule class instance is to be created, and the toolbox icon is the ExampleShapeToolBitmap.

Figure 4 shows the end user view of this toolbox. The toolbox is shown on the left, and the Rule tool is highlighted. A single Rule has been added to this diagram, which initially contains only an instance of the ColorKeyDiag class that describes the possible relations in an AT model and how they are displayed. Relations between model elements are created by selecting a connector tool, e.g. ASDToElement, then clicking on the element that will be at one end of the relation and then the other. The relation builders use the concept of source and target, so the connection tools are named accordingly. That is, an ASDToElement relation is created from a source ASD to a target ASDElement (such as a Rule).



ASD-Element relations are colored in Gainsboro
ASD-Mediation relations are colored in MintCream
Mediation-MediaTING element relations are colored in CornflowerBlue
Mediation-MediaTED element relations are colored in Lavendar
DoL-Community element relations are colored in PaleGreen
Rule-DoL relations are colored in PeachPuff
Tool-DoL relations are colored in ForestGreen
Subject-Community relations are colored in LightSeaGreen
Contradiction relations are colored in DeepPink
Hierarchical decomposition of DoL-to-another-ASD relations are colored in Gold
Networked Outcome-to-Element-in-another-ASD relations are colored in Violet

EDescr	
EMed Type	TING
EName	
ENDecorator	R0
EType	RULE
Fill Color	242, 239, 229
Fill Gradient Mode	Horizontal
Outline Color	113, 111, 110

EDescr
Description for CSU.ActivityTheoryV5.ASDElement.EDescr

Figure 4. End user view of AT DSL Toolbox

AT Language Enumerated Types

Several enumerated types are defined for the language. These are shown in Table 2, along with the domain class and attribute to which they relate.

Table 2. AT Language Enumerated Types

Domain Class	Class Attribute	Enumeration Name	Enumeration Literals
ASDElement	EType	EleTypeENUM	AIM COMM DOL OUTCOME RULE SUBJECT TOOL
ASDElement	EMedType	MedEleTypeENUM	NONE TED TING
Mediation	MType	MedTypeENUM	CA SA SC UNSET
DiagColorKey	InfoType	InfoTypeENUM	AERR DISP
AnalysisError	AEType	AErrorENUM	ASDnoTSAMED MEDnoRels R2DOLS RTDnoMED

			S2COMM T2DOLS TnoSAMED WHODOESDOL
Contradiction	CIType	ConsidItemENUM	ASD ASDELE NOTSET

Diagram Shape Custom Code

Custom code was added for the ASD compartment shape. This shape displays the names of specific types of model elements in the compartments. For example, the ASD shape compartments are Tool, Rule, DoL, Subject, Aim, Community, and Outcome, and the names of all of the ASDElements of each type that are related to the particular ASD are displayed in the associated compartment. Since the metamodel only specifies a relation from ASD to ASDElement, just obtaining the set of related ASDElements is not sufficient to display particular types of these elements in the proper compartment. Therefore a custom code filter was added to take the set of ASDElements related to the ASD, determine if the EType attribute is the desired value (e.g. 'TOOL') and, if so, add the element to a list which is returned after all the related ASDElements have been considered. The names of this filtered list of elements are then displayed in the compartment.

The compartment shape for Contradiction object also displays the names of the items that contradict the considered item, but in this case there are two relations, one for all ASDElements that are contradictory and one for all ASDs that are contradictory, so no filtering needs to occur to get the elements whose names need to be displayed.

Tool Implementation – Visual Studio Solution Projects, Generated and Custom Code

When the DSL definition (metamodel including display elements, any custom domain types, and graphical editor tools) is built by transforming all templates to generate code, two projects are created in the solution: DSL and DSLPackage. Custom code can be added to each of these solutions. The custom code that was added to each solution is discussed below: relation builders, validation methods, and rule code was added to the DSL solution and user command code was added to the DSLPackage solution.

DSL Solution – Relation Builder Custom Code

Relation builders are generated for each relation in the metamodel. These builders check structural constraints as specified by the metamodel, such as whether the object chosen in the graphical editor as the source of the relation is the proper type, and whether any multiplicity constraints on it would be violated if the relation was made. The target of the relation is similarly validated, then the relation is created. There are two methods that can be added to extend the behavior of relation builders using custom code. The first checks the source and the second checks both the source and target. There are currently three sets of relation builders that have associated custom code to enforce constraints beyond the structural ones specified in the metamodel.

The first set of builders has to do with mediations. Structurally a mediation relation from mediation object to mediaTED element must go to a mediaTED element, but the relation builder is enhanced to ensure that only two mediaTED elements are related to the mediation (a source check) and that the two mediaTED elements are of different types (a source and target check) and that the mediation and mediaTED elements are related to the same ASD (also a source and target check). Similarly, a mediation can be related to only one mediaTING element (structural check) and both must be related to the same ASD (a source and target check).

The second set of relation builder methods are used when an Outcome of an ASD is related to an ASD element of a different ASD (a networked relation). In this case the constraint is that the Outcome and the other element must be related to different ASDs, and the target element must not be an Aim or Outcome. Here the check occurs for the source and target.

Finally, the contradiction relation builders also use custom code. The metamodel shows relations from a contradiction to both an ASD considered item and an ASDElement considered item, when, in fact, there can only be one considered item. So the source check is that if the relation for the considered item is being created, the other kind of considered item is not already related to the contradiction. The source and target check is that the target is not part of the set of contradictory items. Similarly, if an ASD or ASDElement is going to be added as a contradictory item, it must not be the considered item (this is a source and target check).

DSL Solution – Validation Custom Code

Validations occur when an AT model is opened, saved, or from the user command menu. There is currently custom code associated with validating two classes – the ATModelRoot and an ASD. Problems found in these methods cause an error to be written to the error window.

The first *ATModelRoot validation method* simply creates and adds a color key information object for the model if it doesn't exist (InfoTypeENUM.DISP in Table 2). The second goes through all contradiction objects in the model and checks that the considered item (an ASD or ASDElement) and the relation to the relevant ASD are consistent. Since the relevant ASD relation can be created or deleted independent of the considered item this check is needed. If the considered item is an ASD then the relevant ASD should be the same ASD. If the considered item is an ASDElement that is only related to one ASD, then the relevant ASD should be one to which it is related.

The *ASD validation methods* include a method to decide whether at least one of each type of ASDElement is related to the ASD (and list any types that are missing in an error message). Another ASD validation is to make sure that any networked element relations do not have the Outcome and related element both being related to this ASD. Another check is that any hierarchical relations between a DoL and another ASD must not include this ASD as part of the hierarchical relationship set. There are four checks that make sure the source of particular relations share this ASD with their targets. The idea is that any Tool to DoL related elements (tools2dols relation) do share this ASD in their sets of related ASDs. The same thing should be true for elements of Rule to DoL relations (rules2dols), Subject to Community relations (subjComm), and DoL to Community relations (whoDoesDoL). There is a problem if the source element is related to multiple ASDs and the target is related to only some of them. In this case, the ASD they share will check out fine, but in another ASD, the relation will still exist from the source element, but the target may not be related to that ASD. This will generate a validation error. The real problem is that each of these relations is valid in a particular context, a particular ASD, and there doesn't seem to be a way to indicate that the relation is valid for only a particular ASD when the elements can be related to many ASDs. Therefore, the convention adopted in these examples is to only relate Community members to multiple ASDs, and to acknowledge that most elements like Tools will perhaps be refined or have detail added and thus be slightly different when they are used in different activities. This problem is discussed in the Issues section of this paper.

DSL Solution – Rules Custom Code

Rules are used to add functionality when elements or relations are created or deleted in the AT tool. There are several rules in the AT tool, outlined in this section and detailed further in the Implementation Details section of this paper.

When a new Tool, Rule, DoL, Subject, Aim, Community, or Outcome element is created, rules are used to set the EType and EMedType attributes. Table 2 shows the respective literals for the EleTypeENUM and MedEleTypeENUM enumerations. The EMedType for an Outcome is set to NONE. When a new Mediation element is created, its MType is set to UNSET (MedTypeENUM in Table 2).

We have found it useful to refer to ASD elements using a short tag name, e.g. 'D4', rather than a descriptor such as 'Planner finds out options'. However, when a model element is first created, its description has not yet been entered. Thus, we can create the short name tag at model element creation time, but we cannot manipulate any part of the description to create a full name for the element. Therefore we created rules that fire on ASDElement or ASD creation to build this short tag name from an overall model counter and then to increment the counter. The counter is an ATModelRoot domain property, and there is also a rule that fires on model creation that sets the counter to 0. (Otherwise the counter just keeps growing every time the model is opened.) The short tag name is stored in one of the ASDElement attributes, and a similar attribute in an ASD. We then added a rule that fires when an ASDElement property is changed, and if the object description property has been changed then a full name for the element is created by taking the short tag name and appending the full element description onto it. This concatenated name is the one that will be displayed as part of the proper ASD shape compartment (e.g. in the 'DoLs' compartment, 'D4: Planner finds out options').

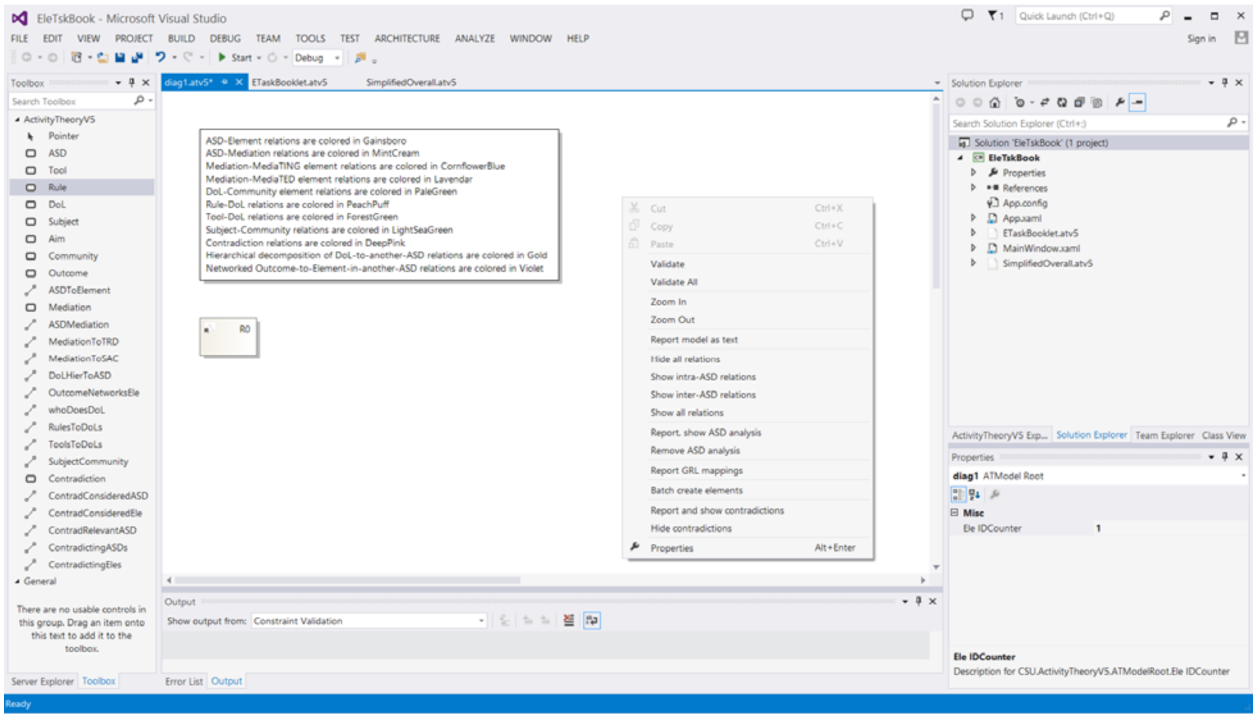
When a Mediation object to mediated object relation is added we invoke a rule that checks to see if this is the second such relation, and if so it sets the proper MType enumeration value (MedTypeENUM in Table 2). If this is the first such relation we cannot set the type, but in both cases we can set an object attribute for the mediated element name.

When a Mediation to mediated object relation is deleted, the MType must be set to UNSET (MedTypeENUM in Table 2), and the associated attribute name needs to be set to blank.

Finally, when a considered item is related to a contradiction, if the relevant ASD relation is not already set then we may be able to set it automatically. If the considered item is an ASD, then the relevant ASD should be the same ASD. If the considered item is an ASDElement that is only related to one ASD, then we can also set the relevant ASD to that same ASD. A design decision was made to not attempt to fix the relevant ASD link if it is already set. Adding this functionality would be complex since this is a relation that can be manipulated independently of the considered item relations.

DSLPackage Solution – User Commands

When a user right-clicks on a blank area of an AT model diagram the user command menu appears. There are many commands that can be added here; we have added them in several groups. An example of this menu is shown in Figure 5. Since no model elements were selected prior to displaying this menu the first three items are not available and are greyed out. They and the two validation-related commands are automatically generated and added to this menu by the DSL SDK.


ASD-Element relations are colored in Gainsboro
ASD-Mediation relations are colored in MintCream
Mediation-MediaTING element relations are colored in CornflowerBlue
Mediation-MediaTED element relations are colored in Lavendar
DoL-Community element relations are colored in PaleGreen
Rule-DoL relations are colored in PeachPuff
Tool-DoL relations are colored in ForestGreen
Subject-Community relations are colored in LightSeaGreen
Contradiction relations are colored in DeepPink
Hierarchical decomposition of DoL-to-another-ASD relations are colored in Gold
Networked Outcome-to-Element-in-another-ASD relations are colored in Violet

There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.

Ele IDCounter
Description for CSU.ActivityTheoryV5.ATModelRoot.Ele IDCounter

Figure 5. User Command Menu

The first group of AT language-specific commands allows the user to zoom in and out when viewing the model diagram. This capability should really be added to the tool bar, but it is not clear how to do this, so it was added as user commands.

The next group causes a textual discription of the model to be created and written to a file. Only ASCII text is written to the file, but it is up to the user to provide an appropriate name and file extension. This is accomplished using a VS SaveFileDialog object. The file is written on a per-ASD basis, grouping the type of ASDElements that are related to the ASD. Additional relations (tools2dols, rules2dols, whoDoesDoL, hierarchical, subjComm, and network relations) are also written to the file. Mediations related to the ASD are included. Finally, any ASDElements that are not related to any ASD are written to the file. Contradictions are not included in this report.

The next group of commands has to do with showing/hiding the many possible relations in the model, and is aimed at improving readability of the diagram. The commands are to hide all relations, show relations in the ASD that are not ASD to ASDElement or ASD to Mediation relations (i.e. show tools2dols, rules2dols, whoDoesDoL, and, subjComm relations), show hierarchical and network relations, and show all relations.

The next group is related to model analyses. One command performs the analyses, modifies the diagram using color to indicate problems, and writes analysis results to a file. The other command removes all analysis information from the diagram and the model store. When an analysis is performed and a problem is found, an AnalysisError object is created with the corresponding error type (AErrorENUM in Table 2), and related to the problem model element (an ASD, ASDElement, or Mediation object). In addition a DiagColorKey object with the type AERR (InfoTypeENUM in Table 2) is created and added to the model if it does not already exist. Currently the following items are checked as part of the analysis:

- Every Rule, Tool, and DoL is involved in a mediation.
- Every ASD has at least one mediation: a Tool mediates between a Subject and an Aim. (This is, after all, the point of an ASD – for a Subject to achieve an Aim using a mediating Tool.)

- Every Tool mediates between a Subject and an Aim.
- Every Mediation is related to two mediated elements and one mediating element.
- Every Rule is related to at least one DoL (rules2DoL).
- Every Tool is related to at least one DoL (tools2DoL)
- Every DoL is related to at least one Community member (whoDoesDoL)
- Every Subject is related to exactly one Community member (subjComm)

The next group of user commands writes proposed mappings to GRL elements to a file, based on the GRL trace-links defined in our previous work. It currently consists of writing details specifying the following suggested steps:

1. Create GRL Actors associated with AT Community member objects if they do not already exist in the GRL model.

2. Create GRL Tasks associated with AT DoLs.

3. Place Tasks in the corresponding Actors based on the whoDoesDoL relation, except in the case where multiple Community member objects are in the whoDoesDoL relation. In that case, the Task should be placed outside the Actors, and decomposed into Tasks that can be placed in each Actor.

4. Create GRL Goals associated with AT Aims and GRL Goals associated with AT Outcomes.

5. Create positive contributions from the Goals associated with Aims to those associated with Outcomes.

6. Create GRL Resources associated with AT Tools.

7. Create dependencies in the GRL model based on mediations between AT Tools and AT Subjects and Aims. This is accomplished by first determining the AT Community member associated with the AT Subject, then the AT DoLs associated with that AT Community member, then the AT Tools associated with each AT DoL, looking for the AT Tool that is involved in the mediation. If one is found, there needs to be a dependency in the GRL Actor associated with the AT Community member that the GRL Task associated with the AT DoL depends on the GRL Resource associated with the AT Tool and that the GRL Goal associated with the AT Aim also depends on this GRL Resource. Since multiple such dependencies may be indicated from the AT model, a note is added that only one such dependency should be added. Also modeler intervention is needed in the case where multiple GRL Tasks or Resources are involved, since not all may be part of a dependency.

8. Create GRL Goals or GRL Softgoals associated with AT Rules. GRL Goals have clear indications of being met whereas GRL Softgoals are used when it may not be clear when they are met.

9. Create a GRL dependency that a GRL Task associated with an AT DoL depends on the GRL Goal or Softgoal associated with the AT Rule involved in the rules2dols relation. Further, if the AT DoL is related to any AT Tools (tools2dol relation), then the associated GRL Resource may depend on the GRL Goal or Softgoal associated with the AT Rule. In this case either or both of the GRL Task and GRL Resource may be depend on the GRL Goal/Softgoal. Modeler intervention is usually needed to make this decision.

10. Create GRL dependencies base on the tools2dols relations in the AT model as follows. Create a GRL dependency that a GRL Task maped from an AT DoL depends on the GRL Resource mapped from an AT Tool if such a relation was not previously created from a Tool-Subject-Aim mediation as described in step 7 above.

Hierarchical and network relations have not yet been added to these steps.

The next group of user commands is a researcher convenience group that creates multiple ASDElement objects of the same type and sets their descriptions to a string specified in a text file. The file must have a single character on the first line indicating what type of ASDElements should be created, and one description on each subsequent line. Elements of the type are created and added to the ATModelRoot. The modeler must make all subsequent relation additions to the elements to relate them to other model elements. This command can often save time when initially developing a model. For example, a file called aim.txt was used to create three Aim objects for use in the second example described in Section 6 below:

```
A
Make arrangements to go on vacation
Provide information
Investigate and choose destinations
```

Finally, the last group of user commands involves contradictions. One commands shows and writes contradiction information to a file, and the other command hides these relations and contradiction objects from the display.

5. Implementation Details

AT Model Validations

The AT DSL includes model validation that occurs when the model is opened or saved or specifically requested via the user command menu. Violations cause errors to be written to the Error window. Validation methods are defined for two classes: ATModelRoot and ASD.

A. The following validations are defined for the overall model, **ATModelRoot** (**Dsl->CustomCode->Validation->ATModelRoot.cs**):

```
…
[ValidationState(ValidationState.Enabled)]
public partial class ATModelRoot

…

// When the model is opened a DiagColorKeys object describing the color coding used
// for model relations is created if it doesn't already exist since the user might have
// explicitly deleted it at some point.
[ValidationMethod(ValidationCategories.Open)]
ModelHasDisplayInfo(ValidationContext context)

…

// When the model is opened or saved, or from the user command menu, all Contradiction
// objects are checked to see if: 1) if the considered item is an ASD, the relevantASD
// relation is set to the same ASD, or 2) if the considered item is an ASDElement and it
// is related to only a single ASD, then the relevantASD relation is set to that same
// ASD. If these relations are not set as in (1) or (2), then the relevantASD relation
// may not be consistent. The modeler must decide if this is a real error or not. This
// check was added since the relevantASD relation can be manipulated independently of
// considered item relation.
[ValidationMethod(ValidationCategories.Open | ValidationCategories.Save | ValidationCategories.Menu)]
ModelContradictionRelevantASDRelationsConsistent(ValidationContext context)
```

B. The following validations are defined for an **ASD** (**Dsl->CustomCode->Validation->ASD.cs**):

```
…
[ValidationState(ValidationState.Enabled)]
public partial class ASD
```

```
…

// On model open, save, or via user command menu, an ASD is checked to see if it has
// at least one of each type of ASDElement. If it doesn’t then it is not a well-formed
// ASD.
[ValidationMethod(ValidationCategories.Open | ValidationCategories.Save | ValidationCategories.Menu)]
ValidateASDhas7Elements(ValidationContext context)

…

// On model open, save, or via user command menu, an ASD is checked to make sure that
// any of its Outcomes that have Network relations are to ASDElements in other ASDs.
 [ValidationMethod(ValidationCategories.Open | ValidationCategories.Save | ValidationCategories.Menu)]
ValidateNetwrkEleOutInDiffASDs(ValidationContext context)

…

// On model open, save, or via user command menu, an ASD is checked to make sure that
// any of its Tools that have tools2dols relations are to DoLs in the same ASD.
 [ValidationMethod(ValidationCategories.Open | ValidationCategories.Save | ValidationCategories.Menu)]
ValidateToolDoLInSameASD(ValidationContext context)

…

// On model open, save, or via user command menu, an ASD is checked to make sure that
// any of its Rules that have rules2dols relations are to DoLs in the same ASD.
 [ValidationMethod(ValidationCategories.Open | ValidationCategories.Save | ValidationCategories.Menu)]
ValidateRuleDoLInSameASD(ValidationContext context)

…

// On model open, save, or via user command menu, an ASD is checked to make sure that
// any of its Subjects that have subjComm relations are to Community objects in the
// same ASD.
 [ValidationMethod(ValidationCategories.Open | ValidationCategories.Save | ValidationCategories.Menu)]
ValidateSubjectCommunityInSameASD(ValidationContext context)

…

// On model open, save, or via user command menu, an ASD is checked to make sure that
// any of its DoLs that have whoDoesDoL relations are to Community objects in the
// same ASD.
 [ValidationMethod(ValidationCategories.Open | ValidationCategories.Save | ValidationCategories.Menu)]
ValidateDoLCommInSameASD(ValidationContext context)

…

// On model open, save, or via user command menu, an ASD is checked to make sure that
// any of its DoLs that have Hierarchical relations are to other ASDs.
 [ValidationMethod(ValidationCategories.Open | ValidationCategories.Save | ValidationCategories.Menu)]
ValidateDoLHierASDDiffASDs(ValidationContext context)
```

AT Model Rules

Hard constraints, or Rules, are enforced when the model is changed, such as when a new domain class instance is added to an AT model. The following **Rules** are defined (**Dsl->CustomCode->Validation->Rules.cs**):

```
NewATModelRoot
NewASD
NewTool
NewRule
NewDivisionOfLabor
NewSubject
NewAim
NewCommunity
```

```
NewOutcome
NewMediation
NewMedToTED
MRefTEDDel
NewContradToCIASD
NewContradToCIAele
ChangeDescr
```

The ‘NewATModelRoot’ rule resets the overall model counter used to create ASD and ASDElement short tag names to 0. The ‘NewASD’ rule generates a short tag name for the ASD and saves is it in an attribute (not currently used).

The ‘New’ rules for ASDElements are used to set the proper attribute type for the element, e.g. ‘DOL’, and the proper attribute for its mediation element type, e.g. ‘TING’ in the case of a RULE, TOOL, or DOL element type. In addition the short tag name (e.g. ‘D4’) is created and saved in an element attribute. The ‘ChangeDescr’ rule fires when an ASDElement description is changed, and it is used to create the full name of the ASD element that will be displayed in the ASD shape compartment, for example ‘D4: Planner finds out options’.

The ‘NewMediation’ rule sets the proper attribute for the mediation type, ‘UNSET’ when a mediation is first created, and via the ‘NewMedToTED’ rule on the second addition of a mediated element relation: a Subject-Aim, Community-Aim, or Subject-Community mediation. This latter rule also sets the mediated element name attributes in the Mediation object. This adding rule and its inverse, the deletion rule ‘MRefTEDDel’, are used to change a decorator icon in the graphical mediation shape that indicates the mediation type.

The Contradiction rules ‘NewContrad..’ are used to set the type of considered element as an attribute in the Contradiction object, and if possible, also set the relevantASD relation. This relation is set only if it is currently null. In the case that the new considered item is an ASD then the relevantASD relation is set to this same ASD. If the new considered item is an ASDElement, and it is only related to one ASD, then the relevantASD relation is also set to this ASD.

Sample code (**Dsl->CustomCode->Coherence->InitEle.cs**) is:

```
…

[RuleOn(typeof(ATModelRoot), FireTime = TimeToFire.TopLevelCommit)]
public sealed class NewATModelRoot : AddRule
{
    public override void ElementAdded(ElementAddedEventArgs e)
    {
        ATModelRoot newRoot = e.ModelElement as ATModelRoot;
        newRoot.EleIDCounter = 0;
    }
}
…
```

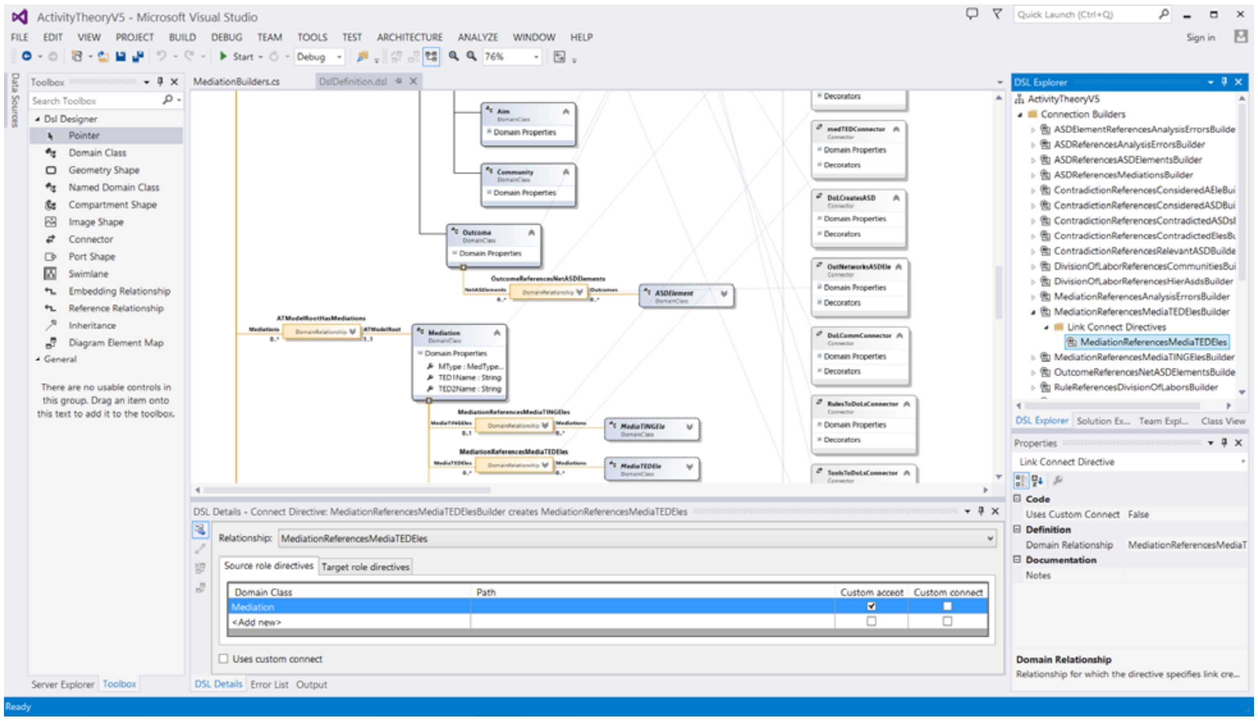
AT Model Relation Builders

In order to ensure that relations are properly created between model elements during rather than just during validation, the builders for the relationships can be enhanced to check that the target and source elements are appropriate. Thus, for example, the builder can enforce that a mediation must be related to two different types of mediated ASDElement. For each relation builder, there are two methods that have to be supplied, both of which return Booleans. The first determines whether a particular object can be the source of the relation, and the second determines whether a particular combination of objects can be the source and target for the relation. The types of the objects and multiplicities given in the metamodel are automatically enforced.

These additional constraints on the **Relation Builders** are found in:

- **Dsl->CustomCode->Validation->MediationBuilders.cs**
- **Dsl->CustomCode->Validation->NetworkOutcomeBuilders.cs**
- **Dsl->CustomCode->Validation->ContradictionBuilders.cs**

To enable the additional methods, first a builder in the DSLExplorer must be selected, and in the DSLmapping tab when this mapping line is selected, the checkbox on the source tab must be marked to specify that the relation has a custom accept directive for the source. Figure 6 shows a screen shot of this action. The highlighted line near the bottom of the window shows the check box for a custom accept of a source for this relation. When the templates are transformed as part of building the DSL, method calls are added to the generated **Dsl->GeneratedCode->ConnectionBuilders.cs** file.



ActivityTheoryV5 - Microsoft Visual Studio
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST ARCHITECTURE ANALYZE WINDOW HELP
Toolbox
Dsl Designer
Pointer
Domain Class
Geometry Shape
Named Domain Class
Compartment Shape
Image Shape
Connector
Port Shape
Swimlane
Embedding Relationship
Reference Relationship
Inheritance
Diagram Element Map
General
MediationBuilders.cs
DslDefinition.dsl
Aim
Community
Outcome
OutcomeReferencesNetASDElements
ASDElement
ATModelRootHasMediations
Mediation
MediationReferencesMediaTINGEles
MediaTINGEle
MediationReferencesMediaTEDEles
MediaTEDEle
medTEDConnector
DoLCreatesASD
OutNetworksASDEle
DoLCommConnector
RulesToDoLsConnector
ToolsToDoLsConnector
DSL Explorer
ActivityTheoryV5
Connection Builders
ASDElementReferencesAnalysisErrorsBuilde
ASDReferencesAnalysisErrorsBuilder
ASDReferencesASDElementsBuilder
ASDReferencesMediationsBuilder
ContradictionReferencesConsideredAEleBui
ContradictionReferencesConsideredASDBui
ContradictionReferencesContradictedASDsI
ContradictionReferencesContradictedElesBu
ContradictionReferencesRelevantASDBuilde
DivisionOfLaborReferencesCommunitiesBui
DivisionOfLaborReferencesHierAsdsBuilder
MediationReferencesAnalysisErrorsBuilder
MediationReferencesMediaTEDElesBuilder
Link Connect Directives
MediationReferencesMediaTEDEles
MediationReferencesMediaTINGElesBuilder
OutcomeReferencesNetASDElementsBuilde
RuleReferencesDivisionOfLaborsBuilder

DSL Details - Connect Directive: MediationReferencesMediaTEDElesBuilder creates MediationReferencesMediaTEDEles

Relationship: MediationReferencesMediaTEDEles

Source role directives | Target role directives

Domain Class	Path	Custom accept	Custom connect
Mediation		☑	☐
<Add new>		☐	☐

☐ Uses custom connect

Properties
Link Connect Directive

Code
Uses Custom Connect False
Definition
Domain Relationship MediationReferencesMediaT
Documentation
Notes

Domain Relationship
Relationship for which the directive specifies link cre...

Figure 6. Screen shot showing that custom builders will be added for Mediation->Mediated Element relations

Example source for the builders of a relation between a Mediation object and a mediated ASDElement object (i.e. Subject, Community, or Aim) is shown below. Note that in all of these methods, deciding whether two elements are in the same or different ASDs is reduced to the problem of deciding whether their sets of related ASDs have a non-empty intersection or not. This is simple in the case of a Mediation since it can only have one related ASD, but more complex in the case of e.g., a networked outcome relation. This problem is a consequence of the general problem of allowing ASDElement objects to be related to multiple ASDs.

```
…
static partial class MediationReferencesMediaTEDElesBuilder
{
   // This method checks that the Mediation is first related to an ASD, and then
   // that it doesn’t already have 2 relations to mediated element objects.
```

```
    private static bool CanAcceptMediationAsSource(Mediation candidate)
    {
       if (candidate.ASD == null)
           return true;
       else
       {
           if (candidate.MediaTEDEles.Count >1)
              return false;
           else
              return true;
    }

    // This method is called after the Mediation was checked as a potential source
    // and the target is a valid mediated element object. What has to be done here
    // is check that the source and target are in the same ASD and if this is the
    // 2nd mediated element object that it and the 1st one are of different types.
    private static bool CanAcceptMediationAndMediaTEDEleAsSourceAndTarget
                      (Mediation sourceMediation, MediaTEDEle targetMediaTEDEle)
    {
       if (sourceMediation.MediaTEDEles.Count == 0)
       {
           if (targetMediaTEDEle.Asds.Contains(sourceMediation.ASD))
               return true;
           else
               return false;
       }
       else
       {
           if (!targetMediaTEDELe.Asds.Contains(sourceMediation.ASD))
               return false;
           else
           {
              EleTypeENUM firstTED = sourceMediation.MediaTEDeles.FirstOrDefault().EType;
              if (targetMediaTEDEle.EType != firstTED)
                  return true;
              else
                  return false;
           }
        }
     }
 }
```

Custom accept methods are also supplied for the **MediationReferencesMediaTINGElesBuilder** to make sure the Mediation object and the mediating element object are both in the same ASD, for the **OutcomeReferencesNetASDElementsBuilder** to make sure the target is not an Aim or Outcome, and that the target and the source are not in the same ASD, and four relations having to do with Contradictions.

The first is **ContradictionReferencesConsideredASDBuilder** where the checks are that there is not already a considered item that is an ASDElement, and that the considered ASD is not a member of the set of contradicted ASDs. The second is **ContradictionReferencesConsideredAEleBuilder** where we have to make sure there is not already a considered item of an ASD, and that the considered ASDElement is not part of the contradicted ASDElements set. The third is **ContradictionReferencesContradictedASDsBuilder** where we must be sure that the new contradicted ASD isn't the considered item, and the fourth is **ContradictionReferencesContradictedEleBuilder** where we have to make sure the new contradicted ASDElement isn't the considered item.

AT Model User Commands

Files associated with the user menu commands are part of the DSLPackage solution. Two generated files must be modified, and additional code added to the solution. The first generated file that must be changed is **SolutionExplorer->DslPackage/Commands.vsct**. This is an XML file that has user command menu information in it. We defined different groups for the commands, then buttons within those groups that will access the command code. New GUID values must be created for each group (accomplished with a built-in VS tool, and of the form "{…}" in the example below), and group identifier and command identifier values given to these items. An excerpt of this file is shown for the zoom in command:

```
…
<CommandTable …>
<Commands package="guidPkg">
    <Groups>
      <Group guid="guidZoom" id="grpidZoom" priority="0x0100">
        <Parent guid="guidCmdSet" id="menuidContext" />
      </Group>
…
    </Groups>
      <Buttons>
        <Button guid="guidZoom" id="cmdidZoomIn" priority="0x0100" type="Button">
          <Parent guid="guidZoom" id="grpidZoom"/>
          <CommandFlag>DynamicVisibility</CommandFlag>
          <Strings>
            <ButtonText>Zoom In</ButtonText>
          </Strings>
        </Button>
…
      </Buttons>
</Commands>
<VisibilityConstraints>
  <VisibilityItem guid="guidZoom" id="cmdidZoomIn" context="guidEditor"/>
…
</VisibilityConstraints>
  <Symbols>
    <!-- Substitute a unique GUID for the placeholder: -->
    <GuidSymbol name="guidZoom" value="{C0F07C5B-CB2A-42A6-B483-56189F817A75}" >
      <IDSymbol name="grpidZoom" value="0x01001"/>
      <IDSymbol name="cmdidZoomIn" value="0x00001"/>
    </GuidSymbol>
…
  </Symbols>
</CommandTable>
```

The next file that must be changed is part of the Generated Code that causes the Commands.vsct file to be read and processed. This file is in the **SolutionExplorer->DSLPackage/GeneratedCode/Package.tt**. The line is described as changing the version of the language (this change was from '1' to '2'):

```
[VSShell::ProvideMenuResource("1000.ctmenu", 2)]
```

Code must be added to set the GUID and identifiers and to augment the list of generated commands. An excerpt of the file **SolutionExplorer->DSLPackage/CustomCode/CommandInfo.cs** dealing with the zoom in command is shown below.

```
…
namespace CSU.ActivityTheoryV5
{
    internal partial class ActivityTheoryV5CommandSet
    {
        private Guid guidZoom = new Guid("C0F07C5B-CB2A-42A6-B483-56189F817A75");
        private const int grpidZoom = 0x01001;
        private const int cmdidZoomIn = 1;
…
        protected override IList<MenuCommand> GetMenuCommands()
        {
            IList<MenuCommand> commands = base.GetMenuCommands();
            DynamicStatusMenuCommand zoomInCmd =
              new DynamicStatusMenuCommand(
                new EventHandler(OnStatusZoomInCmd),
                new EventHandler(OnMenuZoomInCmd),
                new CommandID(guidZoom, cmdidZoomIn));
            commands.Add(zoomInCmd);
…
            return commands;
        }
    }
}
```

Code next needs to be added for the event handlers. The following is an excerpt from the file **SolutionExplorer->DSLPackage/CustomCode/ComdOnInfoZoom.cs** for the zoom in command.

```
…
namespace CSU.ActivityTheoryV5
{
    internal partial class ActivityTheoryV5CommandSet
    {
        private void OnStatusZoomInCmd(object sender, EventArgs e)
        {
            MenuCommand command = sender as MenuCommand;
            command.Visible = command.Enabled = false;
            if (this.IsDiagramSelected())
            {
                command.Visible = true;
                command.Enabled = true;
            }
        }

        private void OnMenuZoomInCmd(object sender, EventArgs e)
        {
            MenuCommand command = sender as MenuCommand;
            Store store = this.CurrentDocData.Store;
            ATModelRoot modelRoot = null;
            ReadOnlyCollection<ModelElement> modelElements =
              store.ElementDirectory.FindElements(ATModelRoot.DomainClassId);
            modelRoot = (ATModelRoot)modelElements[0];
            Char action = 'i';

            ZoomDiag(modelRoot, action);
        }
    }
}
```

Finally, the code to actually perform the command needs to be added. An excerpt from the file **SolutionExplorer->DSLPackage/CustomCode/Zoom.cs** that zooms the diagram is shown below.

```
…
namespace CSU.ActivityTheoryV5
```

```
{
    internal partial class ActivityTheoryV5CommandSet
    {
        private void ZoomDiag(ATModelRoot thRoot, Char inOut)
        {
            ActivityTheoryV5Diagram diag =
              PresentationViewsSubject.GetPresentation(thRoot).FirstOrDefault()
                                                   as ActivityTheoryV5Diagram;
            DiagramView dv = diag.ActiveDiagramView;
            if (inOut == 'i')
                dv.ZoomIn();
            else
                dv.ZoomOut();

        }
    }
}
```

6. Examples

We explored three example systems while developing the AT language, its USE realization, and the Visual Studio AT DSL tool. These are introduced below, then the results of using the VS AT tool to model each of them is discussed in more detail in the following subsections. Where appropriate, particular VS AT tool functions that the examples highlight are also discussed.

We made extensive use of a case study system developed in Mexico by Dr. Lozano-Fuentes of the CSU department of Microbiology, Immunology, and Pathology and others while we explored AT and developed the original AT language metamodel that we realized in the USE tool. This system is a Dengue virus data capture and interpretation system that was developed jointly with CSU and a University and Public Health Ministry in Mexico. The ASDs presented in Section 6.1 describe the context of the vector surveillance portion of this system. Vector surveillance entails looking for potential breeding sites of the mosquito that carries Dengue, *Aedes aegypti*, in residential areas. This activity was the target of a cell phone application that field agents can take to residences and use to directly enter data regarding how many and of what type of breeding sites are found. Data is uploaded by this application to a centralized database where it is available for interpretation and policy decision-making. Policy decisions specifically affect vector control, which may include spraying for adult mosquitos or applying larvicide to breeding sites. The cell phone application was field tested in the city of Mérida, Yucatán [1][8]. The related vector surveillance ASD has a rather low level of abstraction, so its scope is quite limited. It makes a good AT example since it relies on other activities to provide tools essential to achieving it, and also because it was derived from activities that take place at a higher level of abstraction. The overall context of this ASD therefore requires both networked and hierarchical decomposition relations.

The vector surveillance activity is complex, so a second, simpler example was also developed. This example is one of taking a vacation. Planning, coordinating, and traveling on vacation with several people can be time-consuming and a challenge. The example consists of the following scenario. A group of ten or so people is going to take a two-week vacation to some places they have heard about but never been. There are different age groups and generations in the travel group. One adult has been chosen by the other adults to be the overall coordinator and travel planner. Some constraints have been set by the adults; for example an upper bound on price and the fact that the destinations must be new to them. Other guidelines have been voiced, for example, no tour busses, no mega-resorts, some vacation actions everyone can enjoy, others targeted towards specific members of the group, and at least some vacation actions that the group can enjoy in relative isolation without huge crowds of other people. Several persons pass on ideas they have heard about from acquaintances and friends, regarding both places to visit and places that should be avoided. This scenario was constructed to demonstrate network and hierarchical relationships among activities, and also to specifically demonstrate potential contradictions that AT can help discover.

The third example discussed in this paper is one from another case study at CSU. This example applies AT concepts to a series of semi-structured interviews regarding a proposed transition from a paper-based system that keeps track of student skill mastery in a professional degree to an electronic method. The current task booklet requires signature sign-off that a junior or a senior in the program has demonstrated a particular skill deemed necessary for the degree. Completed booklets are required for graduation, and are also used in the accreditation process with the associated professional organization. Due to the sheer volume of information in the interviews, this example brought to light scaling issues with the VS AT tool and especially its mapping to GRL model elements, and resulted in proposed changes to the metamodel, a new user command to quickly create multiple ASDElements, and changes to the GRL mapping user command.

Each of these examples is discussed in more detail in the following sections.

6.1 Vector Surveillance System

The Survey Premises ASD is based on the cell phone application portion of the Dengue system case study. It relies on another activity to provide tools, a coordinator activity that provides lists of premises that need to be surveyed by field agents searching for potential mosquito breeding sites. The network of these two activities is shown in Figure 7 below. The name of the ASD dealing with actual surveillance is Survey Premises, or SP, and the name of the other ASD is Assign Surveillance Tasks to Personnel, or ASTP. We briefly explain the AT concepts shown in the figure; additional information can be found in Appendix A.

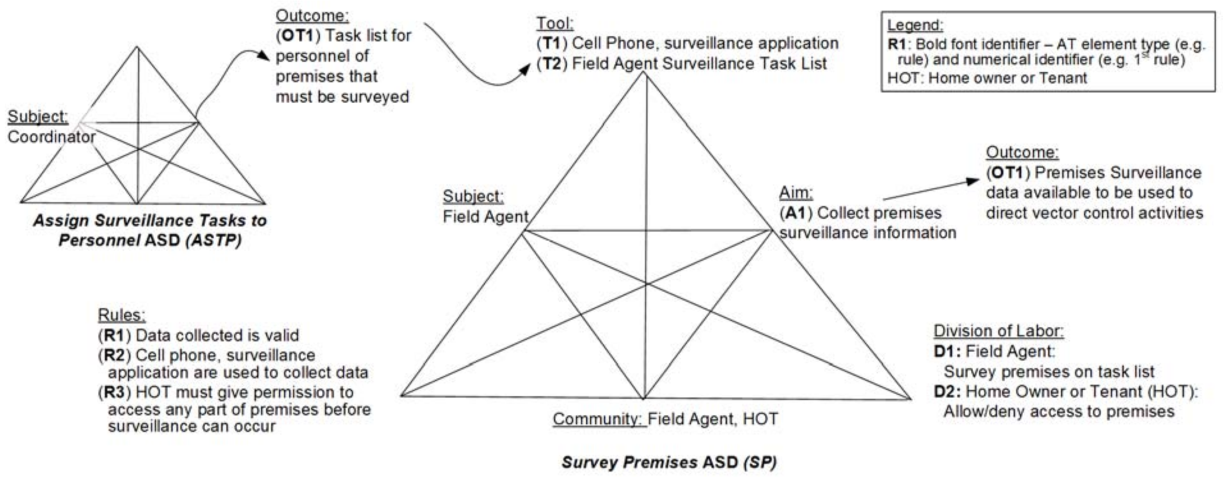

Outcome:
(OT1) Task list for personnel of premises that must be surveyed
Subject: Coordinator
Assign Surveillance Tasks to Personnel ASD (ASTP)
Tool:
(T1) Cell Phone, surveillance application
(T2) Field Agent Surveillance Task List
Legend:
R1: Bold font identifier – AT element type (e.g. rule) and numerical identifier (e.g. 1st rule)
HOT: Home owner or Tenant
Outcome:
(OT1) Premises Surveillance data available to be used to direct vector control activities
Subject: Field Agent
Aim:
(A1) Collect premises surveillance information
Rules:
(R1) Data collected is valid
(R2) Cell phone, surveillance application are used to collect data
(R3) HOT must give permission to access any part of premises before surveillance can occur
Division of Labor:
D1: Field Agent: Survey premises on task list
D2: Home Owner or Tenant (HOT): Allow/deny access to premises
Community: Field Agent, HOT
Survey Premises ASD (SP)

Figure 7. SP and ASTP networked ASDs (Visio tool)

In AT, every activity has one or more Aims that lead to Outcomes as a result of one or more Subjects completing the activity. The activity Aims are shared by members of the Community who are defined as anyone having a vested interest in the Aims. Subjects use Tools to achieve the Aims, and Tools are considered the mediating elements between a Subject and an Aim. Subjects interact with the Community according to explicit and implicit conventions or norms, which are called Rules. Rules are considered the mediating elements between a Subject and the Community members. Finally, the Community breaks up the activity in order to achieve the Aim through Division of Labor (DoL). DoLs are considered the mediating element between Community members and the Aim. These seven elements constitute a well-formed activity system, and we show them in activity system diagrams (ASDs), developed by Engeström[2].

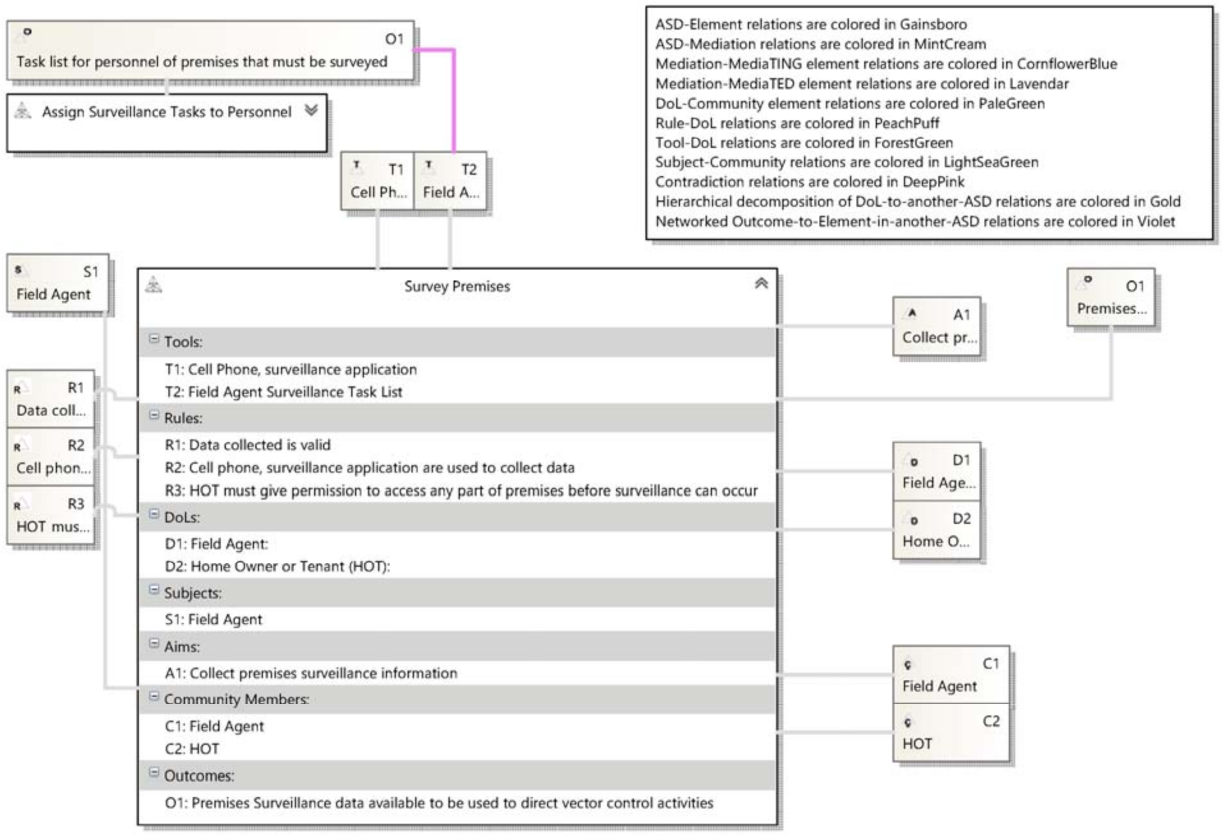
In the Survey Premises (SP) activity, diagrammed in Figure 7, a Field Agent uses a task list (which is created by a Coordinator as part of the ASTP activity) to decide which premises need to be surveyed. The Field Agent uses the cell phone surveillance application on their cell phone to collect information about existing or potential mosquito breeding sites at a particular premises on the list. This is only allowed if the home owner or tenant gives permission for the surveillance. Thus we can identify the Subject (Field Agent), some Tools (task list, cell phone application, and cell phone), another Community member (home owner/tenant), a Rule (home owner/tenant must give permission to access any part of premises before surveillance can occur), and two DoLs (home owner/tenant allows or denies access to premises, and Field Agent surveys premises on task list). Figure 7 shows additional Rules for the SP ASD, and the Outcome of this activity, namely that premises surveillance data is available to be used to determine appropriate vector control actions.

All elements (ASDElement class in the AT language metamodel) of the SP activity are shown in Figure 7, but only the Subject and Outcome elements of the ASTP activity are shown. This Outcome becomes a Tool for the SP activity as is shown by the curved arrow from the Outcome to the Tool. Figure 7 was developed using the Microsoft Visio tool. Changes to the diagram must be made by hand, and none of the ASDElement relations defined in the AT language metamodel such as whoDoesDoL, tools2dols, rules2dols, or subjComm are shown. Mediation relations are not shown, and no contradiction relations are included. The only additional relation shown in this figure is the network relation (enabledEleRefOut in the metamodel), which is shown using a curved arrow from the ASTP Outcome OT1 to the SP Tool T2. It is possible to show hierarchical relations from a DoL in one ASD to another ASD in a similar way, using lines from the appropriate DoL to the name of the hierarchically decomposed ASD.

Figure 8 shows the same network specified using the Visual Studio AT language tool. While the Engeström triangle is not used in the tool, an ASD is shown as a compartment shape with this triangle as an icon in its top left corner. The ASD compartments correspond to the various ASDElements that are related to the ASD. Icons in the same location on the ASDElement shapes show the relative location of the element type on the ASD triangle, although these are very faint in Figure 8. For example, Rules have the following icon: R△ .

ASD shapes have an expand/collapse icon in the upper right corner that can be used to show or hide the contents of the compartments (the icon and functionality are part of the DSL SDK). The ASTP ASD is collapsed, while the SP ASD is expanded. ASDElements are named according to the description that was input when the element was created, and decorated in their upper right corner with a short tag name annotation that indicates the type of element (i.e. 'S' for a Subject), and an increasing integer number. Thus, the SP ASD has three Rule objects, annotated as 'R1', 'R2', and 'R3'. Their size has been left as the default size, but since the ASD is expanded and has been widened, the full name descriptions can be found in the Rule compartment.

A color key for the different relations shown in a model is displayed in the upper right of Figure 8. The network relation from O1, in the ASTP ASD, to T2, in the SP ASD, in Figure 7 above is shown as a Violet colored line in Figure 8, and all of the ASD-to-ASDElement relations are shown in Gainsboro. No additional relations have been added to this model, but the color key shows how they would be shown if they existed.


ASD-Element relations are colored in Gainsboro
ASD-Mediation relations are colored in MintCream
Mediation-MediaTING element relations are colored in CornflowerBlue
Mediation-MediaTED element relations are colored in Lavendar
DoL-Community element relations are colored in PaleGreen
Rule-DoL relations are colored in PeachPuff
Tool-DoL relations are colored in ForestGreen
Subject-Community relations are colored in LightSeaGreen
Contradiction relations are colored in DeepPink
Hierarchical decomposition of DoL-to-another-ASD relations are colored in Gold
Networked Outcome-to-Element-in-another-ASD relations are colored in Violet

O1
Task list for personnel of premises that must be surveyed

Assign Surveillance Tasks to Personnel

Survey Premises

Tools:
T1: Cell Phone, surveillance application
T2: Field Agent Surveillance Task List

Rules:
R1: Data collected is valid
R2: Cell phone, surveillance application are used to collect data
R3: HOT must give permission to access any part of premises before surveillance can occur

DoLs:
D1: Field Agent:
D2: Home Owner or Tenant (HOT):

Subjects:
S1: Field Agent

Aims:
A1: Collect premises surveillance information

Community Members:
C1: Field Agent
C2: HOT

Outcomes:
O1: Premises Surveillance data available to be used to direct vector control activities

Figure 8. SP and ASTP networked ASDs (Visual Studio AT Language tool)

The next example, the vacation example, has been more fully modeled in order to demonstrate other metamodel relations.

6.2 Take a Vacation Example

Following the scenario described at the beginning of this section, several Visio diagrams were created to show different ASDs involved in this example. Figure 9 shows the top level ASD, Take a Vacation.

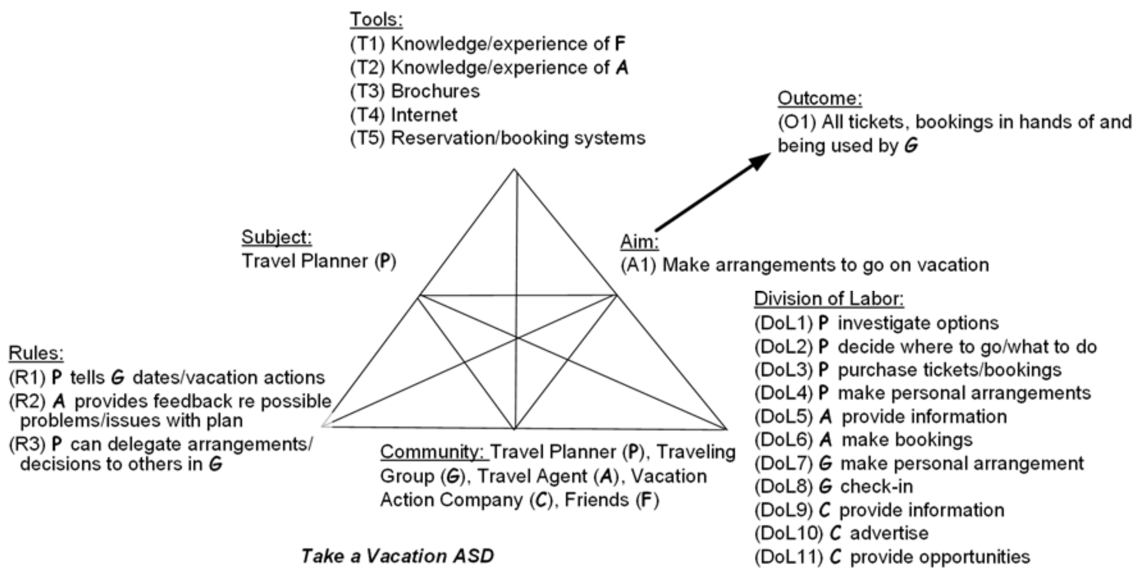

Tools:
(T1) Knowledge/experience of F
(T2) Knowledge/experience of A
(T3) Brochures
(T4) Internet
(T5) Reservation/booking systems
Outcome:
(O1) All tickets, bookings in hands of and being used by G
Subject:
Travel Planner (P)
Aim:
(A1) Make arrangements to go on vacation
Division of Labor:
(DoL1) P investigate options
(DoL2) P decide where to go/what to do
(DoL3) P purchase tickets/bookings
(DoL4) P make personal arrangements
(DoL5) A provide information
(DoL6) A make bookings
(DoL7) G make personal arrangement
(DoL8) G check-in
(DoL9) C provide information
(DoL10) C advertise
(DoL11) C provide opportunities
Rules:
(R1) P tells G dates/vacation actions
(R2) A provides feedback re possible problems/issues with plan
(R3) P can delegate arrangements/ decisions to others in G
Community: Travel Planner (P), Traveling Group (G), Travel Agent (A), Vacation Action Company (C), Friends (F)
Take a Vacation ASD

Figure 9. Top level Take a Vacation ASD (Visio tool)

This ASD can be decomposed hierarchically, as follows: DoL5 and DoL9 are both part of a Provide Information activity, and DoL1 and DoL2 are both part of an Investigate and Decide activity. In addition, this example shows network relations because the Provide Information activity creates tools used in the Investigate and Decide activity. Figure 10 shows the Provide Information ASD. The hierarchical decomposition is not shown in this figure.

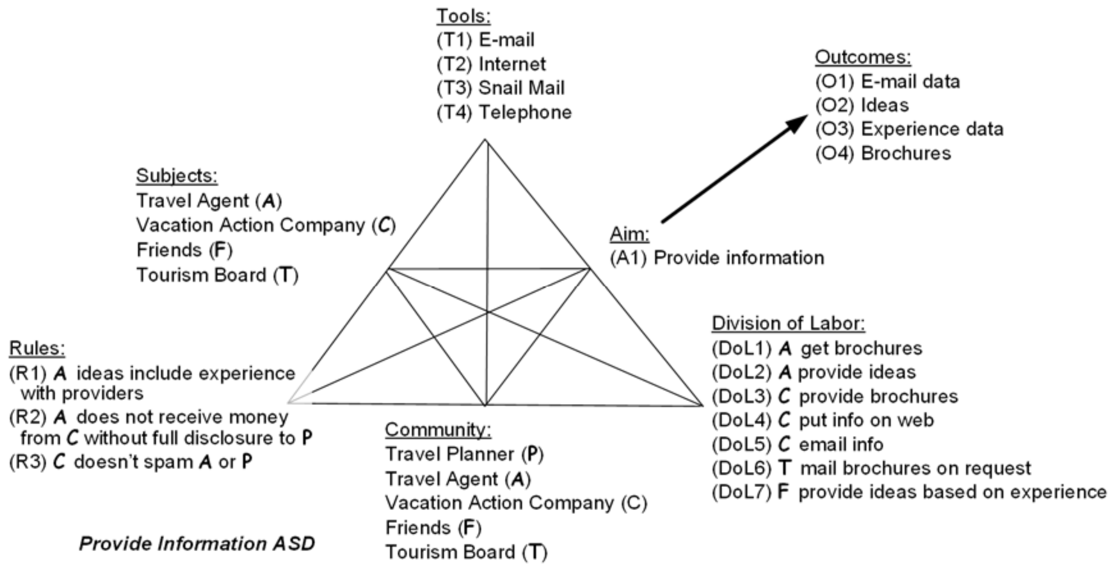


Figure 10. Provide Information ASD, resulting from hierarchical decomposition of DoL5 and DoL9 in Figure 9 (Visio tool)

Figure 11 shows the Investigate and Decide ASD, with the network relations from Outcomes of the Provide Information ASD to Tools in the Investigate and Decide ASD. Again, hierarchical decomposition is not shown in the figure.

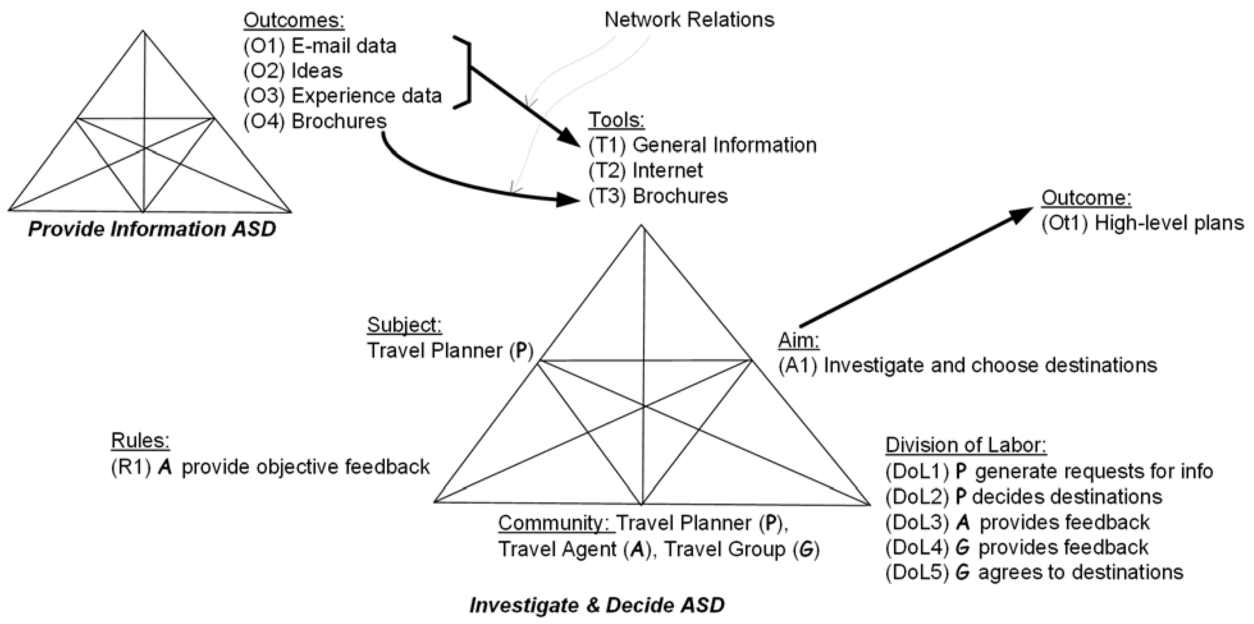
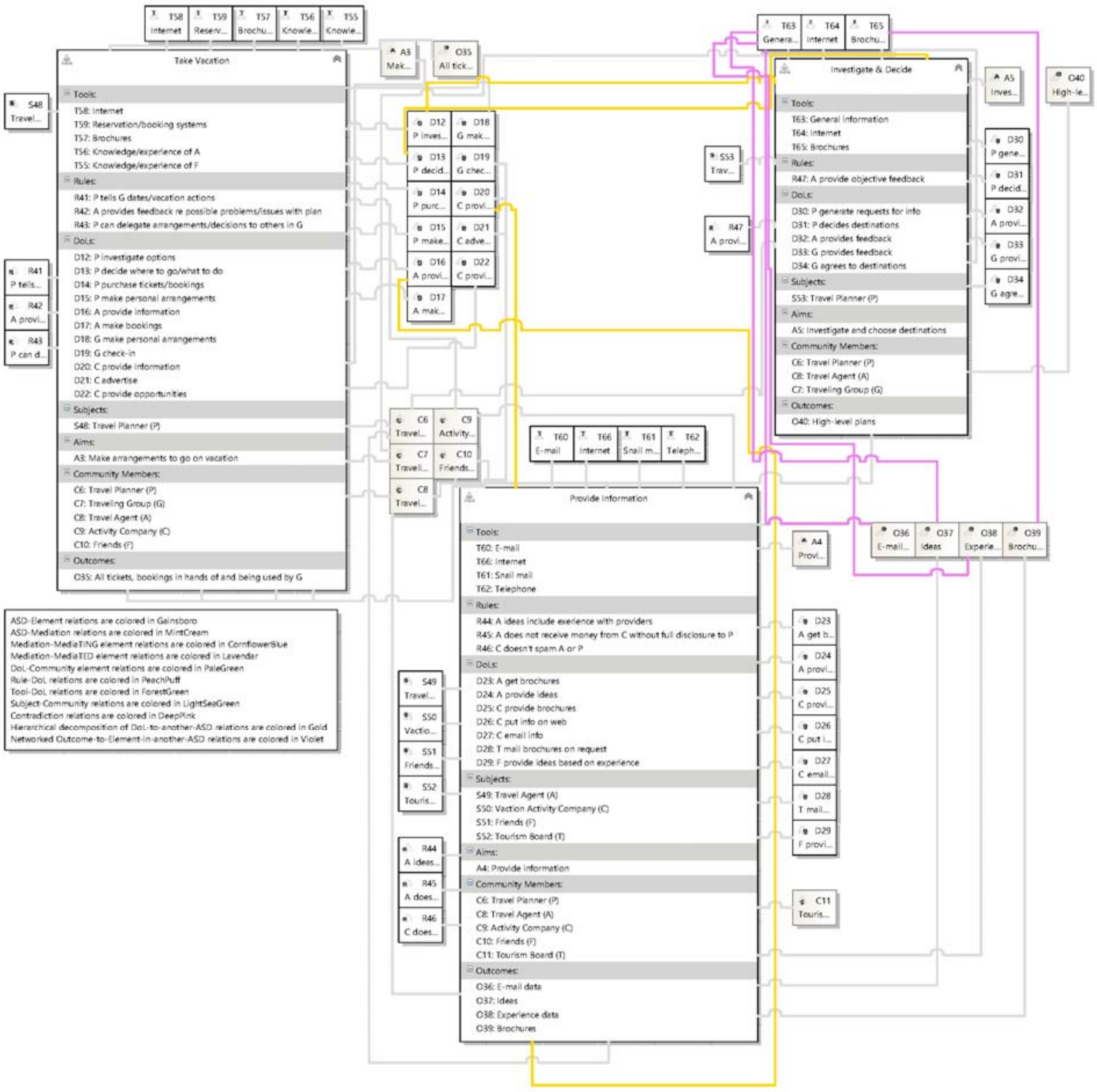

Outcomes:
(O1) E-mail data
(O2) Ideas
(O3) Experience data
(O4) Brochures
Provide Information ASD
Network Relations
Tools:
(T1) General Information
(T2) Internet
(T3) Brochures
Outcome:
(Ot1) High-level plans
Subject:
Travel Planner (P)
Aim:
(A1) Investigate and choose destinations
Rules:
(R1) A provide objective feedback
Division of Labor:
(DoL1) P generate requests for info
(DoL2) P decides destinations
(DoL3) A provides feedback
(DoL4) G provides feedback
(DoL5) G agrees to destinations
Community: Travel Planner (P), Travel Agent (A), Travel Group (G)
Investigate & Decide ASD

Figure 11. Investigate and Decide ASD with network relations from Provide Information ASD (Visio tool)

Similar to Figure 7 all elements of the Investigate and Decide ASD are shown, but only the Outcomes of the Provide Information ASD are shown. The specific Tools that are created as a result of the Provide Information Outcomes are indicated using the arrows labeled “Network Relations”.

These three ASDs were specified using the Visual Studio AT Language tool, and the resulting model is shown in Figure 12. This figure shows the hierarchical decomposition relations (Gold colored lines) in addition to specific Outcome network relationships (Violet colored lines).



Take Vacation

Tools:
T58: Internet
T59: Reservation/booking systems
T57: Brochures
T56: Knowledge/experience of A
T55: Knowledge/experience of F

Rules:
R41: P tells G dates/vacation actions
R42: A provides feedback re possible problems/issues with plan
R43: P can delegate arrangements/decisions to others in G

DoLs:
D12: P investigate options
D13: P decide where to go/what to do
D14: P purchase tickets/bookings
D15: P make personal arrangements
D16: A provide information
D17: A make bookings
D18: G make personal arrangements
D19: G check-in
D20: C provide information
D21: C advertise
D22: C provide opportunities

Subjects:
S48: Travel Planner (P)

Aims:
A3: Make arrangements to go on vacation

Community Members:
C6: Travel Planner (P)
C7: Traveling Group (G)
C8: Travel Agent (A)
C9: Activity Company (C)
C10: Friends (F)

Outcomes:
O35: All tickets, bookings in hands of and being used by G

Investigate & Decide

Tools:
T63: General information
T64: Internet
T65: Brochures

Rules:
R47: A provide objective feedback

DoLs:
D30: P generate requests for info
D31: P decides destinations
D32: A provides feedback
D33: G provides feedback
D34: G agrees to destinations

Subjects:
S53: Travel Planner (P)

Aims:
A5: Investigate and choose destinations

Community Members:
C6: Travel Planner (P)
C8: Travel Agent (A)
C7: Traveling Group (G)

Outcomes:
O40: High-level plans

Provide Information

Tools:
T60: E-mail
T66: Internet
T61: Snail mail
T62: Telephone

Rules:
R44: A ideas include exerience with providers
R45: A does not receive money from C without full disclosure to P
R46: C doesn't spam A or P

DoLs:
D23: A get brochures
D24: A provide ideas
D25: C provide brochures
D26: C put info on web
D27: C email info
D28: T mail brochures on request
D29: F provide ideas based on experience

Subjects:
S49: Travel Agent (A)
S50: Vaction Activity Company (C)
S51: Friends (F)
S52: Tourism Board (T)

Aims:
A4: Provide information

Community Members:
C6: Travel Planner (P)
C8: Travel Agent (A)
C9: Activity Company (C)
C10: Friends (F)
C11: Tourism Board (T)

Outcomes:
O36: E-mail data
O37: Ideas
O38: Experience data
O39: Brochures

ASD-Element relations are colored in Gainsboro
ASD-Mediation relations are colored in MintCream
Mediation-MediaTING element relations are colored in CornflowerBlue
Mediation-MediaTED element relations are colored in Lavendar
DoL-Community element relations are colored in PaleGreen
Rule-DoL relations are colored in PeachPuff
Tool-DoL relations are colored in ForestGreen
Subject-Community relations are colored in LightSeaGreen
Contradiction relations are colored in DeepPink
Hierarchical decomposition of DoL-to-another-ASD relations are colored in Gold
Networked Outcome-to-Element-in-another-ASD relations are colored in Violet

Figure 12. Vacation AT Model (Visual Studio AT Language tool)

Examples of mediations and contradictions for the Take a Vacation ASD and the other ASDElement relations for the Investigate and Decide ASD are shown in Figure 13.

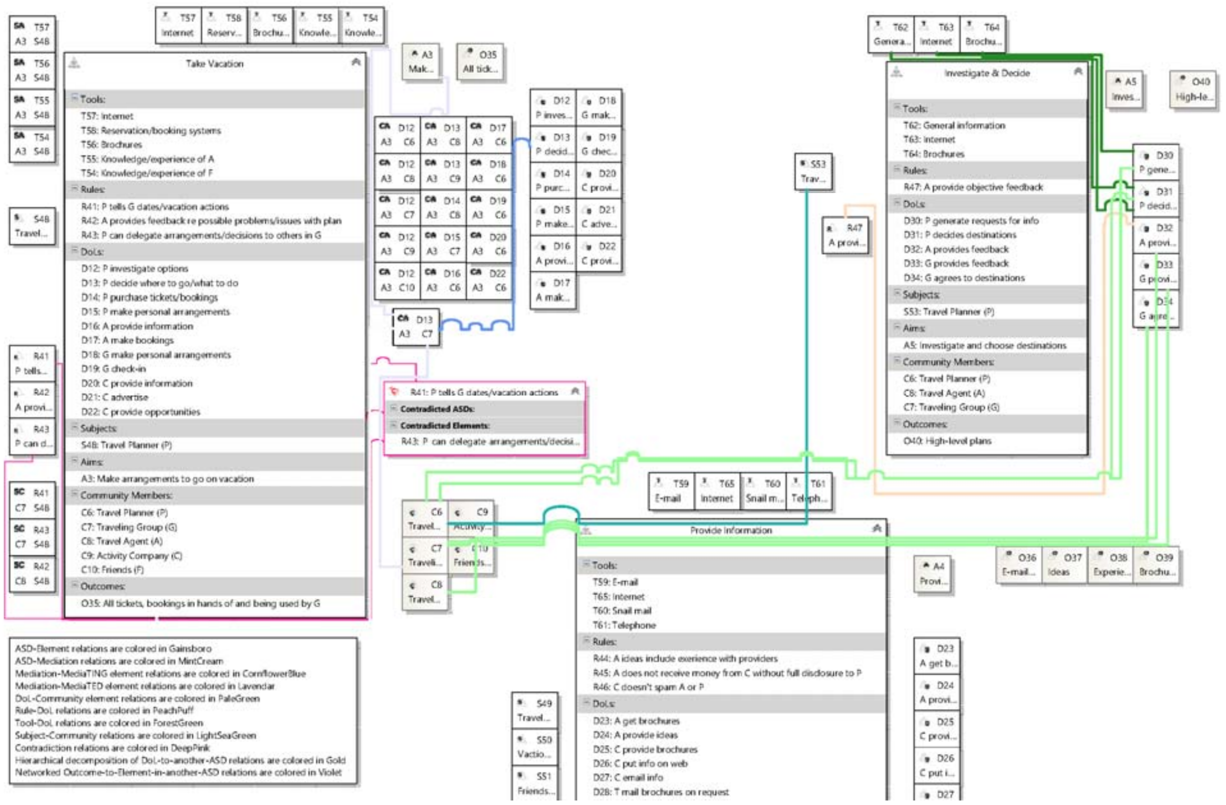

Take Vacation
Tools:
T57: Internet
T58: Reservation/booking systems
T56: Brochures
T55: Knowledge/experience of A
T54: Knowledge/experience of F
Rules:
R41: P tells G dates/vacation actions
R42: A provides feedback re possible problems/issues with plan
R43: P can delegate arrangements/decisions to others in G
DoLs:
D12: P investigate options
D13: P decide where to go/what to do
D14: P purchase tickets/bookings
D15: P make personal arrangements
D16: A provide information
D17: A make bookings
D18: G make personal arrangements
D19: G check-in
D20: C provide information
D21: C advertise
D22: C provide opportunities
Subjects:
S48: Travel Planner (P)
Aims:
A3: Make arrangements to go on vacation
Community Members:
C6: Travel Planner (P)
C7: Traveling Group (G)
C8: Travel Agent (A)
C9: Activity Company (C)
C10: Friends (F)
Outcomes:
O35: All tickets, bookings in hands of and being used by G
R41: P tells G dates/vacation actions
Contradicted ASDs:
Contradicted Elements:
R43: P can delegate arrangements/decisi...
Investigate & Decide
Tools:
T62: General information
T63: Internet
T64: Brochures
Rules:
R47: A provide objective feedback
DoLs:
D30: P generate requests for info
D31: P decides destinations
D32: A provides feedback
D33: G provides feedback
D34: G agrees to destinations
Subjects:
S53: Travel Planner (P)
Aims:
A5: Investigate and choose destinations
Community Members:
C6: Travel Planner (P)
C8: Travel Agent (A)
C7: Traveling Group (G)
Outcomes:
O40: High-level plans
Provide Information
Tools:
T59: E-mail
T65: Internet
T60: Snail mail
T61: Telephone
Rules:
R44: A ideas include experience with providers
R45: A does not receive money from C without full disclosure to P
R46: C doesn't spam A or P
DoLs:
D23: A get brochures
D24: A provide ideas
D25: C provide brochures
D26: C put info on web
D27: C email info
D28: T mail brochures on request
ASD-Element relations are colored in Gainsboro
ASD-Mediation relations are colored in MintCream
Mediation-MediaTING element relations are colored in CornflowerBlue
Mediation-MediaTED element relations are colored in Lavendar
DoL-Community element relations are colored in PaleGreen
Rule-DoL relations are colored in PeachPuff
Tool-DoL relations are colored in ForestGreen
Subject-Community relations are colored in LightSeaGreen
Contradiction relations are colored in DeepPink
Hierarchical decomposition of DoL-to-another-ASD relations are colored in Gold
Networked Outcome-to-Element-in-another-ASD relations are colored in Violet

Figure 13. Mediations and Contradictions for Take a Vacation ASD and additional relations for Investigate and Decide ASD.

Figure 13 shows a set of mediations for the top-level Take Vacation ASD. Each Mediation object is annotated as follows. In the upper left corner, an icon indicating the types of the mediated elements is shown, e.g. SA for a mediation between a Subject and Aim. In the upper right corner of the mediation object the short tag name for the mediating element is shown, e.g. T57 for the upper left-most mediation object in Figure 13. In the lower left corner of the mediation object the short tag name of one of the mediated elements is displayed, and the short name for the other is shown in the lower right corner, e.g. A3 and S48 respectively for the upper left-most mediation object shown in Figure 13. All of the relations have been hidden except for those of one mediation related to this ASD, located to the right of the ASD shape, just below the block of CA-type Mediation objects. This is a Community-Aim mediation where D13 mediates between A3 and C7: Division of Labor (D13) 'P decide where to go/what to do' mediates between the Aim (A3) 'Make arrangements to go on vacation' and the Community member (C7) 'Traveling Group (G)'.

A contradiction between two Rules has also been specified for this ASD, just below the CA-type Mediation object discussed above. Contradiction objects are colored in Deep Pink and are decorated with an icon in the upper left corner ( ).Contradictions are defined as being between a considered item (ASD or ASDElement) and the items that contradict it (other ASDs or ASDElements). A contradiction shape is a compartment shape, so lists of the ASDs that contradict the considered item and/or lists of ASDElements that contradict it can easily be shown. For the example shown in Figure 13, the contradiction is relatively simple – if we consider R41, 'P tells G dates/vacation actions', then R43, 'P can delegate arrangements/decisions to others in G' may be a contradiction of this Rule. The Deep Pink dotted line to the Take a Vacation ASD in Figure 13 denotes that it is the relevant ASD for this contradiction. If the item being considered is an ASD, or it is an ASDElement that is related to only one ASD, this link can be set automatically when the contradiction is related to the considered item. Otherwise it must be set manually by the modeler. The relevant ASD relation is used when contradictions are reported using the corresponding user command menu button. An example of a portion of the contradiction report is shown below:

```
Contradictions for which ASD Take Vacation is the relevant ASD:
     Considered Item is ASD Element: R41: P tells G dates/vacation
actions
          with contradictory elements: R43: P can delegate
arrangements/decisions to others in G
```

Other metamodel relations are shown for the Investigate and Decide ASD on the right side of Figure 13. R47 is related to D32 by the rules2dols relation (PeachPuff colored line). S53 is related to C6 by the subjComm relation (LightSeaGreen line). D30 and D31 are related to C6 by the whoDoesDoL relation (PaleGreen colored lines), D32 is related to C8 by this same relation, and D33 and D34 are related to C7 by this relation. T62 and T64 are related to D31 by the tools2dols relation (ForestGreen lines), and T63 is related to D30 by this same relation.

Finally we show some of the analyses results for this AT model. The result of requesting an analysis using the user command produces the diagram changes shown in Figure 14. (Note that prior to running the analysis two other user commands were run to simplify the diagram by hiding all relations and contradiction information.)

Mediations are very important in Activity Theory, and the only ASD where we added any mediation information is the top-level Take Vacation ASD, so we will discuss mediation analysis errors with respect to this ASD. According to the color key related to analysis errors, any Rule, Tool, or DoL outlined in Red is not involved in any mediation. This can indicate that the model is underspecified in that there are missing mediated elements, or that it is incomplete in that the appropriate relations simply have not be added, or that it is overspecified in that the Rule, Tool, or DoL does not belong in the particular ASD. Further, any Tool with a thick outline does not mediate between a Subject and Aim. We see from Figure 14 that T58, a reservation/booking system is not involved in any mediation (outlined in Red), and therefore it does not mediate between a Subject and Aim (thick outline). In fact, this tool may not belong in this particular ASD since it is a tool often used by a Travel Agent.

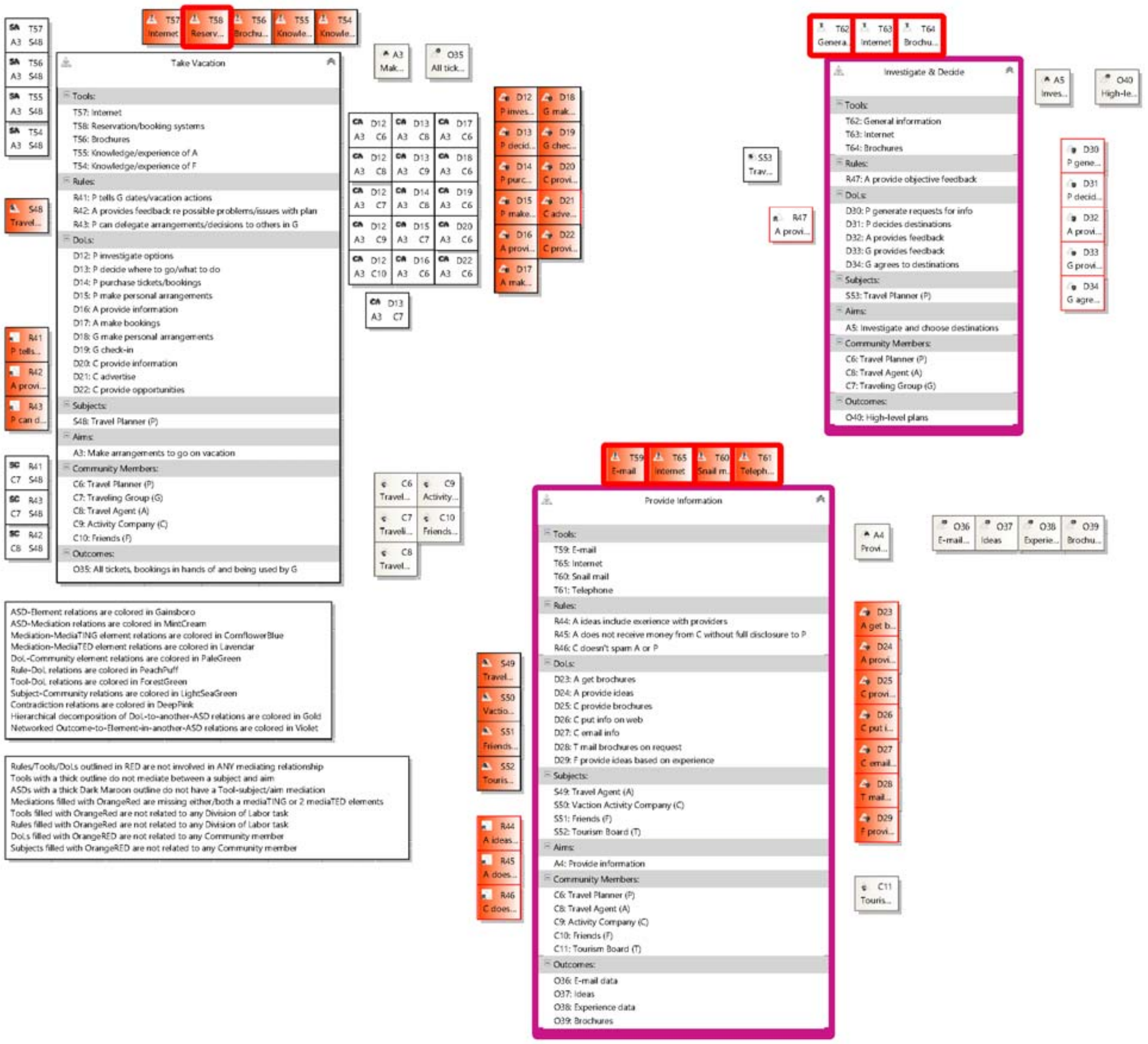
The other Tools in the top-level Take Vacation ASD are all part of Subject-Aim mediations, however, none of them are related to any DoL via the tools2dols relation so they are filled with Orange-Red. The Subject, Rules and all DoLs related to this ASD are also filled with Orange-Red. They are missing the subjComm, rules2dols, and whoDoesDoL relations respectively. Finally, D21 (‘C advertise’) is also outlined in Red, indicating that it does not participate in any mediation. We could argue that it is way to fulfill D20 (‘C provide information’) and therefore that it is an overspecification and can be removed.

Recall that we added rules2dols, tools2dols, whoDoesDoL, and subjComm relations to the Investigate and Decide ASD, so the associated sources of these relations do not show any errors (i.e. no Orange-Red fill color). However, we did not add any mediations to this ASD, so every mediating element is outlined in Red. The Tools all have thick lines since they are not involved in a Subject-Aim mediation, and the ASD has a thick Dark Maroon outline since it has no Tool-SA mediation.

Finally, none of the rules2dols, tools2dols, whoDoesDoL, subjComm, or mediation relations were added to the Provide Information ASD, so all of the mediating elements are outlined in Red, and they and the Subjects are all filled with Orange-Red, and the Tools have thick outlines. The ASD also has a thick Dark Maroon outline.

A report is also created by the analyze user command. It lists analysis errors pertaining to the model and is written to a file specified through a dialog box (the class and associated functionality are available from system packages that can be included in the C# custom code for the user command). A sample of the information in this file is shown below:

```
Relationship Analysis for ASD Take Vacation:
MediaTING elements not involved in any Mediation:
      D21: C advertise
      T58: Reservation/booking systems
Tools that do not mediate a Subject and Aim:
      T58: Reservation/booking systems
Rules not associated with a DoL:
      R41: P tells G dates/vacation actions
…
```


Take Vacation
Tools:
T57: Internet
T58: Reservation/booking systems
T56: Brochures
T55: Knowledge/experience of A
T54: Knowledge/experience of F
Rules:
R41: P tells G dates/vacation actions
R42: A provides feedback re possible problems/issues with plan
R43: P can delegate arrangements/decisions to others in G
DoLs:
D12: P investigate options
D13: P decide where to go/what to do
D14: P purchase tickets/bookings
D15: P make personal arrangements
D16: A provide information
D17: A make bookings
D18: G make personal arrangements
D19: G check-in
D20: C provide information
D21: C advertise
D22: C provide opportunities
Subjects:
S48: Travel Planner (P)
Aims:
A3: Make arrangements to go on vacation
Community Members:
C6: Travel Planner (P)
C7: Traveling Group (G)
C8: Travel Agent (A)
C9: Activity Company (C)
C10: Friends (F)
Outcomes:
O35: All tickets, bookings in hands of and being used by G

Investigate & Decide
Tools:
T62: General information
T63: Internet
T64: Brochures
Rules:
R47: A provide objective feedback
DoLs:
D30: P generate requests for info
D31: P decides destinations
D32: A provides feedback
D33: G provides feedback
D34: G agrees to destinations
Subjects:
S53: Travel Planner (P)
Aims:
A5: Investigate and choose destinations
Community Members:
C6: Travel Planner (P)
C8: Travel Agent (A)
C7: Traveling Group (G)
Outcomes:
O40: High-level plans

Provide Information
Tools:
T59: E-mail
T65: Internet
T60: Snail mail
T61: Telephone
Rules:
R44: A ideas include exerience with providers
R45: A does not receive money from C without full disclosure to P
R46: C doesn't spam A or P
DoLs:
D23: A get brochures
D24: A provide ideas
D25: C provide brochures
D26: C put info on web
D27: C email info
D28: T mail brochures on request
D29: F provide ideas based on experience
Subjects:
S49: Travel Agent (A)
S50: Vaction Activity Company (C)
S51: Friends (F)
S52: Tourism Board (T)
Aims:
A4: Provide information
Community Members:
C6: Travel Planner (P)
C8: Travel Agent (A)
C9: Activity Company (C)
C10: Friends (F)
C11: Tourism Board (T)
Outcomes:
O36: E-mail data
O37: Ideas
O38: Experience data
O39: Brochures

ASD-Element relations are colored in Gainsboro
ASD-Mediation relations are colored in MintCream
Mediation-MediaTING element relations are colored in CornflowerBlue
Mediation-MediaTED element relations are colored in Lavendar
DoL-Community element relations are colored in PaleGreen
Rule-DoL relations are colored in PeachPuff
Tool-DoL relations are colored in ForestGreen
Subject-Community relations are colored in LightSeaGreen
Contradiction relations are colored in DeepPink
Hierarchical decomposition of DoL-to-another-ASD relations are colored in Gold
Networked Outcome-to-Element-in-another-ASD relations are colored in Violet

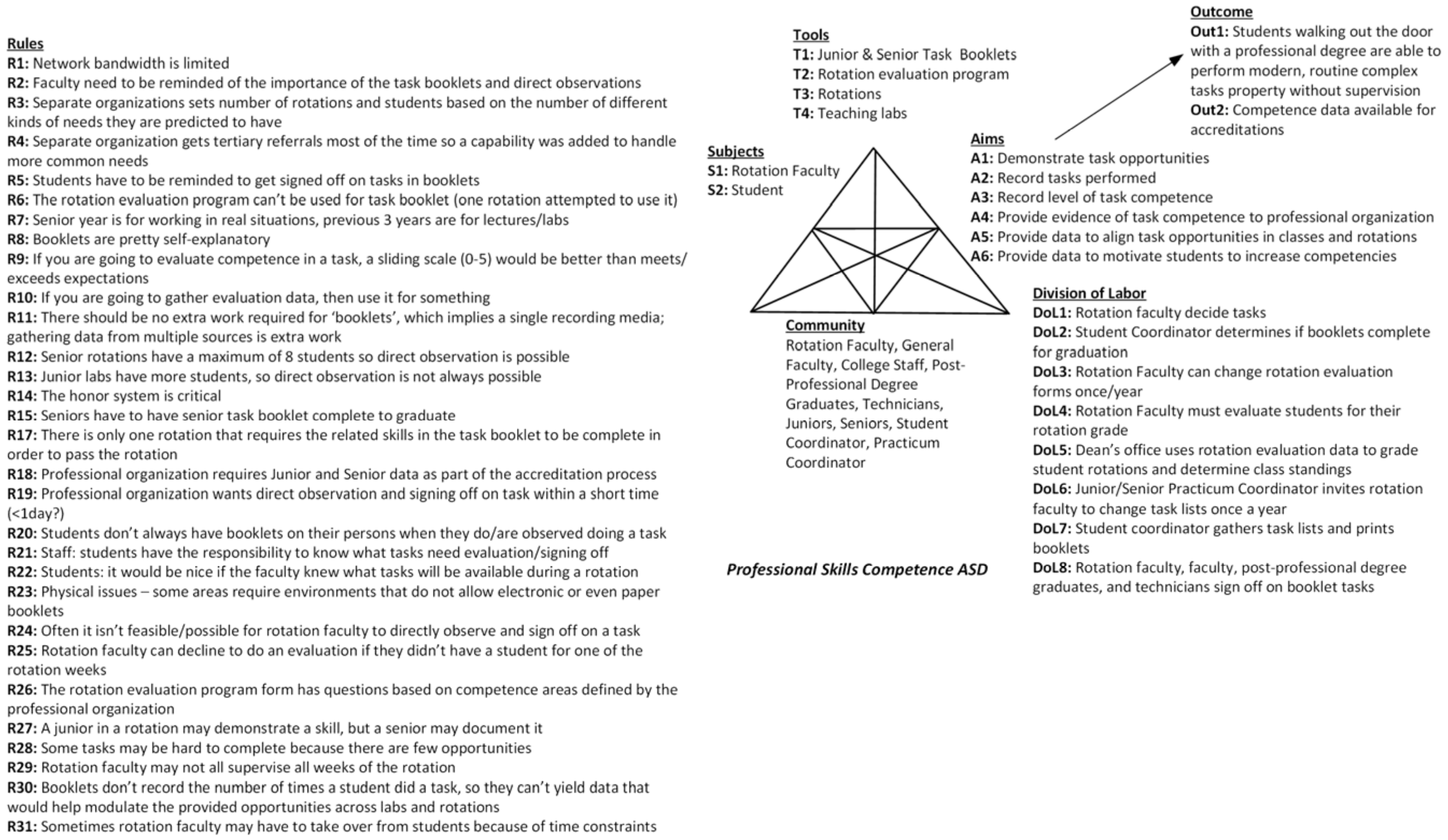
Rules/Tools/DoLs outlined in RED are not involved in ANY mediating relationship
Tools with a thick outline do not mediate between a subject and aim
ASDs with a thick Dark Maroon outline do not have a Tool-subject/aim mediation
Mediations filled with OrangeRed are missing either/both a mediaTING or 2 mediaTED elements
Tools filled with OrangeRed are not related to any Division of Labor task
Rules filled with OrangeRed are not related to any Division of Labor task
DoLs filled with OrangeRED are not related to any Community member
Subjects filled with OrangeRED are not related to any Community member

Figure 14. Analysis diagram of the Vacation ASD network.

6.3 Electronic Task Booklet Investigation Example

This example began with a series of semi-structured interviews aimed at identifying requirements for a proposed electronic version of current paper-based skills evaluation tracking for a professional degree at CSU. The paper task booklets consist of a series of skills and space for a professor to initial agreement that the student has properly demonstrated the skill. There are two booklets, one each for juniors and seniors in the program. The skills are grouped according to lab classes where applicable, and by rotation topics where applicable. Rotations are targeted for seniors, but juniors can also take them. An ASD derived from the interviews is shown in Figure 15.

The interviews were conducted by the author of this Technical Report and Dr. Jaime Ruiz, a professor of Human Computing Interaction in the Computer Science department at CSU. The author developed the initial ASD model.



Rules
R1: Network bandwidth is limited
R2: Faculty need to be reminded of the importance of the task booklets and direct observations
R3: Separate organizations sets number of rotations and students based on the number of different kinds of needs they are predicted to have
R4: Separate organization gets tertiary referrals most of the time so a capability was added to handle more common needs
R5: Students have to be reminded to get signed off on tasks in booklets
R6: The rotation evaluation program can't be used for task booklet (one rotation attempted to use it)
R7: Senior year is for working in real situations, previous 3 years are for lectures/labs
R8: Booklets are pretty self-explanatory
R9: If you are going to evaluate competence in a task, a sliding scale (0-5) would be better than meets/exceeds expectations
R10: If you are going to gather evaluation data, then use it for something
R11: There should be no extra work required for 'booklets', which implies a single recording media; gathering data from multiple sources is extra work
R12: Senior rotations have a maximum of 8 students so direct observation is possible
R13: Junior labs have more students, so direct observation is not always possible
R14: The honor system is critical
R15: Seniors have to have senior task booklet complete to graduate
R17: There is only one rotation that requires the related skills in the task booklet to be complete in order to pass the rotation
R18: Professional organization requires Junior and Senior data as part of the accreditation process
R19: Professional organization wants direct observation and signing off on task within a short time (<1day?)
R20: Students don't always have booklets on their persons when they do/are observed doing a task
R21: Staff: students have the responsibility to know what tasks need evaluation/signing off
R22: Students: it would be nice if the faculty knew what tasks will be available during a rotation
R23: Physical issues – some areas require environments that do not allow electronic or even paper booklets
R24: Often it isn't feasible/possible for rotation faculty to directly observe and sign off on a task
R25: Rotation faculty can decline to do an evaluation if they didn't have a student for one of the rotation weeks
R26: The rotation evaluation program form has questions based on competence areas defined by the professional organization
R27: A junior in a rotation may demonstrate a skill, but a senior may document it
R28: Some tasks may be hard to complete because there are few opportunities
R29: Rotation faculty may not all supervise all weeks of the rotation
R30: Booklets don't record the number of times a student did a task, so they can't yield data that would help modulate the provided opportunities across labs and rotations
R31: Sometimes rotation faculty may have to take over from students because of time constraints

Tools
T1: Junior & Senior Task Booklets
T2: Rotation evaluation program
T3: Rotations
T4: Teaching labs

Subjects
S1: Rotation Faculty
S2: Student

Outcome
Out1: Students walking out the door with a professional degree are able to perform modern, routine complex tasks property without supervision
Out2: Competence data available for accreditations

Aims
A1: Demonstrate task opportunities
A2: Record tasks performed
A3: Record level of task competence
A4: Provide evidence of task competence to professional organization
A5: Provide data to align task opportunities in classes and rotations
A6: Provide data to motivate students to increase competencies

Community
Rotation Faculty, General Faculty, College Staff, Post-Professional Degree Graduates, Technicians, Juniors, Seniors, Student Coordinator, Practicum Coordinator

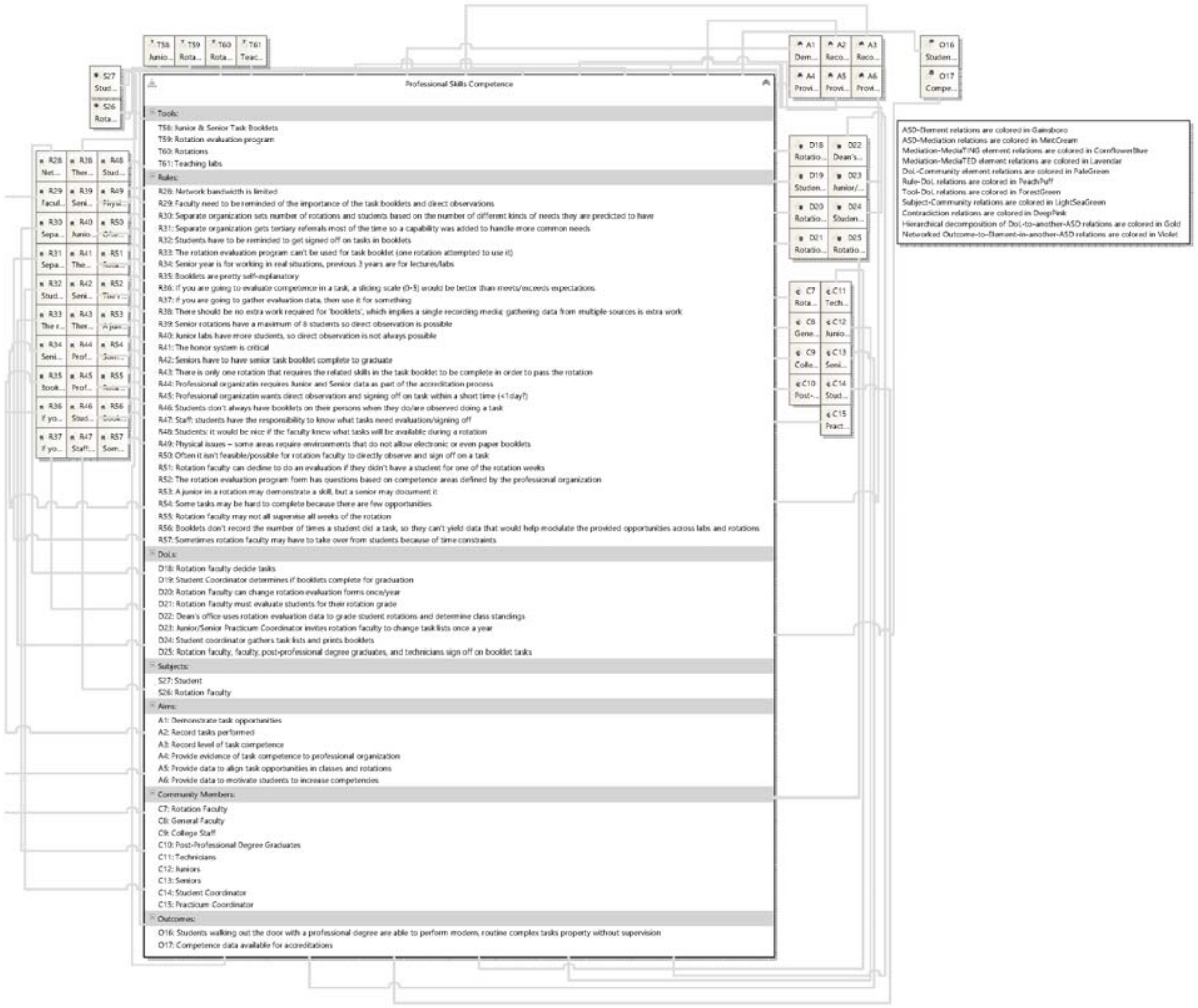
Division of Labor
DoL1: Rotation faculty decide tasks
DoL2: Student Coordinator determines if booklets complete for graduation
DoL3: Rotation Faculty can change rotation evaluation forms once/year
DoL4: Rotation Faculty must evaluate students for their rotation grade
DoL5: Dean's office uses rotation evaluation data to grade student rotations and determine class standings
DoL6: Junior/Senior Practicum Coordinator invites rotation faculty to change task lists once a year
DoL7: Student coordinator gathers task lists and prints booklets
DoL8: Rotation faculty, faculty, post-professional degree graduates, and technicians sign off on booklet tasks

Professional Skills Competence ASD

Figure 15. Initial ASD concerning proposed electronic task booklet (Visio tool)

As can be seen from Figure 15, there are many, many Rules that were found in the interview data. Since this is an initial model, it is also clear that substantial changes would be made to the model as it was discussed with the project stakeholders, so it was specified using the Visual Studio tool to take advantage of automation where possible, to take advantage of the ability to add relations between elements, and to use the analyses available in the tool. In addition, since we were interested in requirements for an electronic task booklet, we wanted to utilize the AT-GRL trace-link mappings to create Goal models to discuss with stakeholders.

The AT model elements were created in the Visual Studio AT tool using the user command that creates multiple elements of the same type from a list of descriptions. This command was added to the VS AT tool as a result of this example. The resulting initial model is shown in Figure 16. This model does not contain any mediation or other element-to-element relations (e.g. tools2dosl).


Professional Skills Competence

Tools:
T58: Junior & Senior Task Booklets
T59: Rotation evaluation program
T60: Rotations
T61: Teaching labs

Rules:
R28: Network bandwidth is limited
R29: Faculty need to be reminded of the importance of the task booklets and direct observations
R30: Separate organization sets number of rotations and students based on the number of different kinds of needs they are predicted to have
R31: Separate organization gets tertiary referrals most of the time so a capability was added to handle more common needs
R32: Students have to be reminded to get signed off on tasks in booklets
R33: The rotation evaluation program can't be used for task booklet (one rotation attempted to use it)
R34: Senior year is for working in real situations, previous 3 years are for lectures/labs
R35: Booklets are pretty self-explanatory
R36: If you are going to evaluate competence in a task, a sliding scale (0-5) would be better than meets/exceeds expectations
R37: If you are going to gather evaluation data, then use it for something
R38: There should be no extra work required for 'booklets', which implies a single recording media; gathering data from multiple sources is extra work
R39: Senior rotations have a maximum of 8 students so direct observation is possible
R40: Junior labs have more students, so direct observation is not always possible
R41: The honor system is critical
R42: Seniors have to have senior task booklet complete to graduate
R43: There is only one rotation that requires the related skills in the task booklet to be complete in order to pass the rotation
R44: Professional organizatin requires Junior and Senior data as part of the accreditation process
R45: Professional organizatin wants direct observation and signing off on task within a short time (<1day?)
R46: Students don't always have booklets on their persons when they do/are observed doing a task
R47: Staff: students have the responsibility to know what tasks need evaluation/signing off
R48: Students: it would be nice if the faculty knew what tasks will be available during a rotation
R49: Physical issues – some areas require environments that do not allow electronic or even paper booklets
R50: Often it isn't feasible/possible for rotation faculty to directly observe and sign off on a task
R51: Rotation faculty can decline to do an evaluation if they didn't have a student for one of the rotation weeks
R52: The rotation evaluation program form has questions based on competence areas defined by the professional organization
R53: A junior in a rotation may demonstrate a skill, but a senior may document it
R54: Some tasks may be hard to complete because there are few opportunities
R55: Rotation faculty may not all supervise all weeks of the rotation
R56: Booklets don't record the number of times a student did a task, so they can't yield data that would help modulate the provided opportunities across labs and rotations
R57: Sometimes rotation faculty may have to take over from students because of time constraints

DoLs:
D18: Rotation faculty decide tasks
D19: Student Coordinator determines if booklets complete for graduation
D20: Rotation Faculty can change rotation evaluation forms once/year
D21: Rotation Faculty must evaluate students for their rotation grade
D22: Dean's office uses rotation evaluation data to grade student rotations and determine class standings
D23: Junior/Senior Practicum Coordinator invites rotation faculty to change task lists once a year
D24: Student coordinator gathers task lists and prints booklets
D25: Rotation faculty, faculty, post-professional degree graduates, and technicians sign off on booklet tasks

Subjects:
S27: Student
S26: Rotation Faculty

Aims:
A1: Demonstrate task opportunities
A2: Record tasks performed
A3: Record level of task competence
A4: Provide evidence of task competence to professional organization
A5: Provide data to align task opportunities in classes and rotations
A6: Provide data to motivate students to increase competencies

Community Members:
C7: Rotation Faculty
C8: General Faculty
C9: College Staff
C10: Post-Professional Degree Graduates
C11: Technicians
C12: Juniors
C13: Seniors
C14: Student Coordinator
C15: Practicum Coordinator

Outcomes:
O16: Students walking out the door with a professional degree are able to perform modern, routine complex tasks property without supervision
O17: Competence data available for accreditations

ASD-Element relations are colored in Gainsboro
ASD-Mediation relations are colored in MintCream
Mediation-MediaTING element relations are colored in CornflowerBlue
Mediation-MediaTED element relations are colored in Lavendar
DoL-Community element relations are colored in PaleGreen
Rule-DoL relations are colored in PeachPuff
Tool-DoL relations are colored in ForestGreen
Subject-Community relations are colored in LightSeaGreen
Contradiction relations are colored in DeepPink
Hierarchical decomposition of DoL-to-another-ASD relations are colored in Gold
Networked Outcome-to-Element-in-another-ASD relations are colored in Violet

Figure 16. Initial AT model for proposed electronic task booklet (Visual Studio AT Language tool)

We used the GRL mapping user command to generate a text file indicating GRL elements that should be created to create a corresponding Goal model. A portion of this file is shown below.

```
1. Create the following Actors if they don't already exist:

      Rotation Faculty
      General Faculty
      College Staff
      Post-Professional Degree Graduates
      Technicians
      Juniors
      Seniors
      Student Coordinator
      Practicum Coordinator

2. Create the following Tasks:
      Rotation faculty decide tasks
      Student Coordinator determines if booklets complete for
graduation
      Rotation Faculty can change rotation evaluation forms once/year
      Rotation Faculty must evaluate students for their rotation grade
      Dean’s office uses rotation evaluation data to grade student
rotations and determine class standings
      Junior/Senior Practicum Coordinator invites rotation faculty to
change task lists once a year
      Student coordinator gathers task lists and prints booklets
      Rotation faculty, faculty, post-professional degree graduates,
and technicians sign off on booklet tasks
…
4. Create the following Goals:
      (derived from Aim) Demonstrate task opportunities
      (derived from Aim) Record tasks performed
      (derived from Aim) Record level of task competence
      (derived from Aim) Provide evidence of task competence to
professional organization
      (derived from Aim) Provide data to align task opportunities in
classes and rotations
      (derived from Aim) Provide data to motivate students to increase
competencies
      (derived from Outcome )Students walking out the door with a
professional degree are able to perform modern, routine complex tasks
property without supervision
      (derived from Outcome )Competence data available for
accreditations

5. Create positive contributions from Goals associated with Aims to
Goals associated with Outcomes
…
```

It was evident from this file that there is probably too much information for the Goal model to be understandable. The ASD elements were therefore combined when they were similar (based on the interview data) and then categorized, with the resulting ASD shown in Figure 17. All of the extra information was retained in text files and is not shown in the figure. The specific abstraction notes for this example are included in Appendix C of this report.

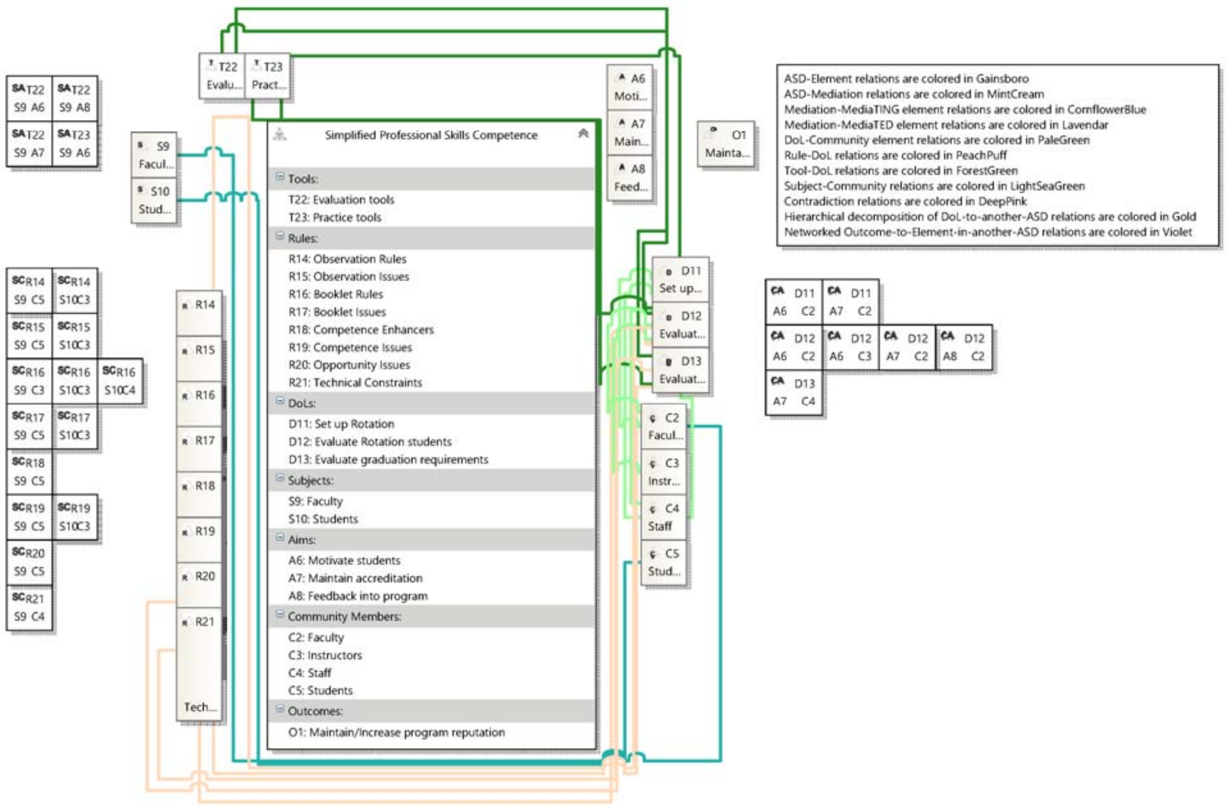

SA T22 S9 A6
SA T22 S9 A8
SA T22 S9 A7
SA T23 S9 A6
T22 Evalu...
T23 Pract...
S9 Facul...
S10 Stud...
A6 Moti...
A7 Main...
A8 Feed...
O1 Mainta...
Simplified Professional Skills Competence
Tools:
T22: Evaluation tools
T23: Practice tools
Rules:
R14: Observation Rules
R15: Observation Issues
R16: Booklet Rules
R17: Booklet Issues
R18: Competence Enhancers
R19: Competence Issues
R20: Opportunity Issues
R21: Technical Constraints
DoLs:
D11: Set up Rotation
D12: Evaluate Rotation students
D13: Evaluate graduation requirements
Subjects:
S9: Faculty
S10: Students
Aims:
A6: Motivate students
A7: Maintain accreditation
A8: Feedback into program
Community Members:
C2: Faculty
C3: Instructors
C4: Staff
C5: Students
Outcomes:
O1: Maintain/Increase program reputation
ASD-Element relations are colored in Gainsboro
ASD-Mediation relations are colored in MintCream
Mediation-MediaTING element relations are colored in CornflowerBlue
Mediation-MediaTED element relations are colored in Lavendar
DoL-Community element relations are colored in PaleGreen
Rule-DoL relations are colored in PeachPuff
Tool-DoL relations are colored in ForestGreen
Subject-Community relations are colored in LightSeaGreen
Contradiction relations are colored in DeepPink
Hierarchical decomposition of DoL-to-another-ASD relations are colored in Gold
Networked Outcome-to-Element-in-another-ASD relations are colored in Violet
SC R14 S9 C5
SC R14 S10C3
SC R15 S9 C5
SC R15 S10C3
SC R16 S9 C3
SC R16 S10C3
SC R16 S10C4
SC R17 S9 C5
SC R17 S10C3
SC R18 S9 C5
SC R19 S9 C5
SC R19 S10C3
SC R20 S9 C5
SC R21 S9 C4
R14
R15
R16
R17
R18
R19
R20
R21
Tech...
D11 Set up...
D12 Evaluat...
D13 Evaluat...
C2 Facul...
C3 Instr...
C4 Staff
C5 Stud...
CA D11 A6 C2
CA D11 A7 C2
CA D12 A6 C2
CA D12 A6 C3
CA D12 A7 C2
CA D12 A8 C2
CA D13 A7 C4

Figure 17. Simplified AT model for proposed electronic task booklet

Only the intra-ASD relations are shown in Figure 17 (mediation and ASD-to-ASDElement relations have been hidden). When the analyze user command is run, there are no analysis issues with this ASD. The user menu command to create suggested GRL mappings uses the ASDElements and information related to the intra-ASD relations to propose a set of steps to create an initial Goal model from this ASD. Examples of this output are shown and discussed below.

```
(ASD: Simplified Professional Skills Competence)
1. Create the following Actors if they don't already exist:
      Faculty
      Instructors
      Staff
      Students

2. Create the following Tasks:
      Set up Rotation
      Evaluate Rotation students
      Evaluate graduation requirements

3. Place the following Tasks inside the corresponding Actors:
      Task 'Set up Rotation' inside Actor 'Faculty'
      Task 'Evaluate Rotation students' inside Actor 'Faculty'
      Task 'Evaluate Rotation students' inside Actor 'Instructors'
      Task 'Evaluate graduation requirements' inside Actor 'Staff'
      Task 'Set up Rotation' inside Actor 'Staff'
      Task 'Evaluate Rotation students' inside Actor 'Staff'
```

```
If there are multiple Actors associated with the same task create the task
outside the Actors and create decomposition tasks for it in the associated
Actors
```

The note associated with Step 3 was added as a result of working with this example, because it is not possible to place the same GRL Task in multiple GRL Actor spheres. We therefore added the directive to decompose the GRL Task into separate tasks for each GRL Actor. The original DoLs (Figure 16) that we abstracted into the DoLs shown in Figure 17 contain this decomposition information. Both D11 and D12 (from Figure 17) have multiple Community members as targets of the whoDoesDoL relation. We therefore created multiple tasks as follows:

D11: Set up Rotation – Two GRL Tasks are created, one in the Faculty Actor, 'Decide tasks' and one in the Staff Actor, 'Invite task changes'.

D12: Evaluate Rotation students – Three GRL Tasks are created, one in the Faculty Actor, 'Faculty evaluate students' one in the Instructor Actor, 'Observe students', and one in the Staff Actor, 'Determine grades'.

An intermediate form of the goal model created in the jUCMNav tool for these three steps is shown in Figure 18.

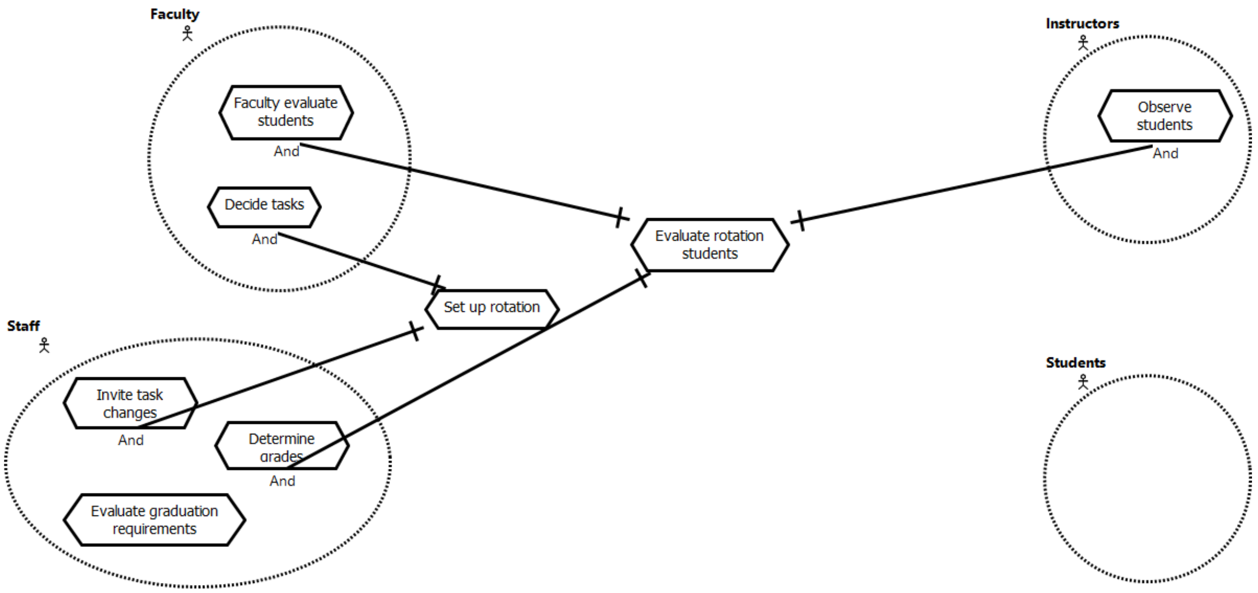


Figure 18. Initial jUCMNav goal model created from first 3 steps of GRL mapping report

Figure 18 shows the decomposition of the tasks with multiple Actors involved in them. In this case a decomposition relation has been created, FROM the Task associated with the DoL (e.g. 'Set up rotation') TO the sub-task located in the Actor (e.g. 'Decide tasks' in the Faculty Actor). The decomposition is an AND decomposition: both sub-tasks must be completed in order for the original task to be completed.

The GRL mapping output continues below:

```
4. Create the following Goals:
      (derived from Aim) Motivate students
      (derived from Aim) Maintain accreditation
      (derived from Aim) Feedback into program
      (derived from Outcome )Maintain/Increase program reputation

5. Create positive contributions from Goals associated with Aims to Goals
associated with Outcomes
```

The intermediate goal model that includes steps 4 and 5 is shown in Figure 19.

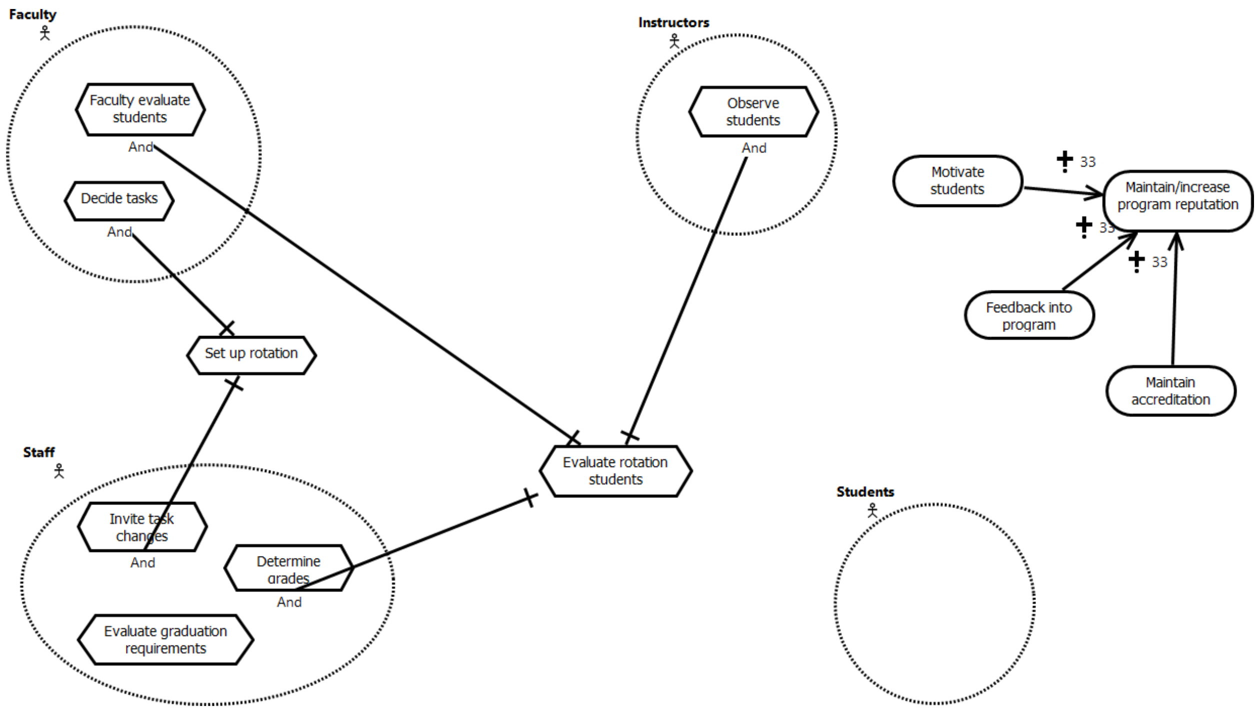


Figure 19. Intermediate Goal model with the addition of Goals from AT Aims and Outcomes.

The Goals derived from Aims are shown to contribute positively (i.e. help) the Goal derived from the Outcome. For now, the relative importance of meeting the contributing Goals is set as equal among them and the assumption denoted by the numerical contributions is that only these three Goals need to be met for the other Goal to be met. Clearly these assumptions may change as modeling progresses.

The GRL mapping continues:

```
6. Create the following Resources:
      Evaluation tools
      Practice tools
```

We next use Tool mediations to suggest dependencies in the Goal model. However, there is an issue relating to creating dependencies based on Tools mediating between Subjects and Aims. The mediating Tool is mapped to a GRL Resource, and Aims are mapped to GRL Goals. Subjects (through the subjComm relation to a Community member) are mapped to GRL Actors. However, an Actor cannot depend on a Resource directly, rather a GRL Task belonging to the Actor must be dependent on the Resource to complete the Task. The problem then is determining what DoLs might be associated with the Subject that use the Tool and then presenting the modeler with this information: one or more of the GRL Tasks associated with the DoLs may be dependent on the GRL Resource associated with the Tool, in order to achieve the GRL Goal associated with the Aim element of the mediation.

The tools2dols relation may be used to propose candidate DoLs and therefore the related GRL Tasks. The Subject of the mediation is related to a Community member via the subjComm relation, and the member may be related to any number of DoLs via the whoDoesDoL relation. The DoLs may in turn then be related to any number of Tools (which hopefully includes the Tool mediating element of the mediation) via the tools2dols relation. All of these relations are used to create the following step.

```
7. Create the following dependencies related to mediation by tools between
subjects and aims:
Only ONE dependency relation should be made between any Goals and Resources
listed in this section
```

```
      Create a dependency that Task 'Evaluate Rotation students' which
resides in Actor 'Faculty' depends on Resource 'Evaluation tools', and
therefore Goal 'Motivate students' also depends on this Resource
…
      Create a dependency that Task 'Set up Rotation' which resides in Actor
'Faculty' depends on Resource 'Practice tools', and therefore Goal 'Motivate
students' also depends on this Resource
…
      If there are multiple Tasks or Resources, then not all of them may be
part of a dependency.
```

While Step 6 is relatively straightforward, Step 7 is not since we decomposed 'Set up Rotation' and 'Evaluate Rotation students' into multiple sub-tasks. The directions in the step must be applied to the sub-tasks that reside in the described Actor. The result of these steps is shown in Figure 20.

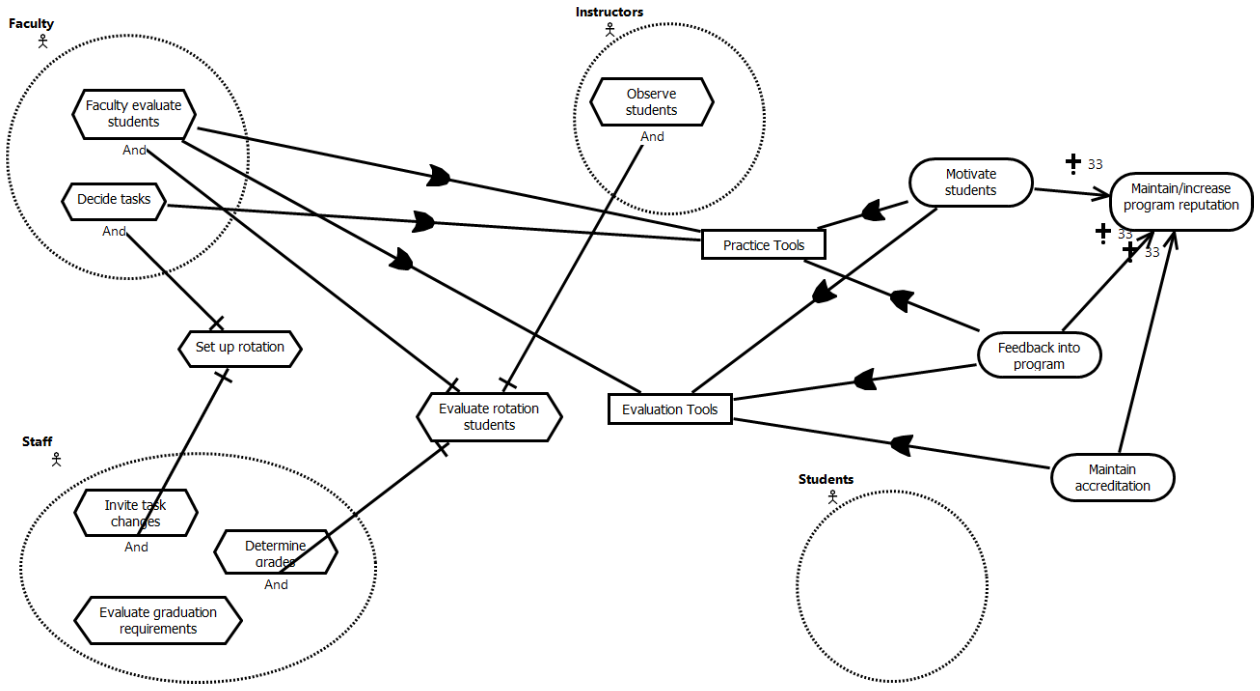


Figure 20. Intermediate Goal model with Resources and some dependencies.

The GRL mapping continues:

```
8. Create the Goals (has clear indications of being met) or Softgoals for the
following (from Rules):
      Observation Rules
      Observation Issues
      Booklet Rules
      Booklet Issues
      Competence Enhancers
      Competence Issues
      Opportunity Issues
      Technical Constraints
```

After studying the detailed Rules used to create the eight Rules in Figure 17, we decided that Goals could be created for 'Observation Issues', 'Booklet Rules' and 'Booklet Issues', 'Competence Enhancers', and 'Opportunity Issues'. Softgoals were created for the other Rules.

Next dependencies of the GRL Goal/Softgoals mapped from AT Rules are considered. Our initial interpretation of Goals and Softgoals created from AT Rules included only the rules2dols relation.

However, while an AT Rule should be related to an AT DoL, it may be the case that the Rule is only applicable to a certain Tool that is used to accomplish the DoL. We discovered this issue with this example, because some Rules pertain to Tools that are used as part of a DoL related to the Rule. A primary example of this situation is the Rule 'R16:Booklet Rules' which has a rules2dols relation to DoL 'D12: Evaluate Rotation students', and yet, these rules pertain to the use of a Tool 'T22: Evaluation Tools'. We therefore modified the AT language metamodel to include a new relation, dols2tools. The code to suggest GRL mappings can then use the rules2dols relation to find pertinent DoLs, then search the DoLs' tools2dols relations to find Tools that might be the true target of the Rule. We considered adding a direct relation between Rules and DoLs, but it seems that some Rules may be related only to the use of a particular Tool under particular circumstances (i.e. DoLs) and not in others. The result of this change to the VS AT language metamodel allows additional steps to be suggested. Examples are shown in step 9.

```
9. Create the following dependencies related to Rules associated with DoLs:
      Create a dependency that Task 'Evaluate Rotation students' depends on
the Goal/Softgoal related to 'Observation Rules'
      Note that Resource 'Evaluation tools' is related to this Task and
therefore the Resource may depend on the Goal/Softgoal(s) listed above
      Note that Resource 'Practice tools' is related to this Task and
therefore the Resource may depend on the Goal/Softgoal(s) listed above
```

After studying the complete set of Rules used to create this Goal/Softgoal, and the complete set of Tools used to create these Resources, we decided that the dependency is from the Task associated with the DoL to the Softgoal associated with the Rule.

```
      Create a dependency that Task 'Evaluate Rotation students' depends on
the Goal/Softgoal related to 'Booklet Rules'
      Note that Resource 'Evaluation tools' is related to this Task and
therefore the Resource may depend on the Goal/Softgoal(s) listed above
      Note that Resource 'Practice tools' is related to this Task and
therefore the Resource may depend on the Goal/Softgoal(s) listed above
```

After studying the complete set of Rules used to create this Goal/Softgoal, and the complete set of Tools used to create these Resources, we decided that the dependency is from the Resource 'Evaluation Tools' to the Goal associated with the Rule.

Next dependencies on Resources (mapped from Tools) from Tasks (mapped from DoLs) that are based on the dols2tools relation can be added. An example from the GRL mapping output is shown below:

```
10. Create the following dependencies related to Tools associated with DoLs:
      If this Tool has an SA mediation that was described in #7 above (the
mapped DoL below already has a dependency on the mapped Tool) then do not
create the dependency.
      If the DoL was decomposed when it was mapped to a Task, then the either
the Task or just some of the sub-Tasks may depend on the Resource mapped from
the Tool.
      Create a dependency that the Task mapped from DoL 'Evaluate Rotation
students' depends on the Resource mapped from Tool 'Evaluation tools'
```

Figure 21 shows the addition of all the Rule-related Softgoals/Goals and their dependency relations.

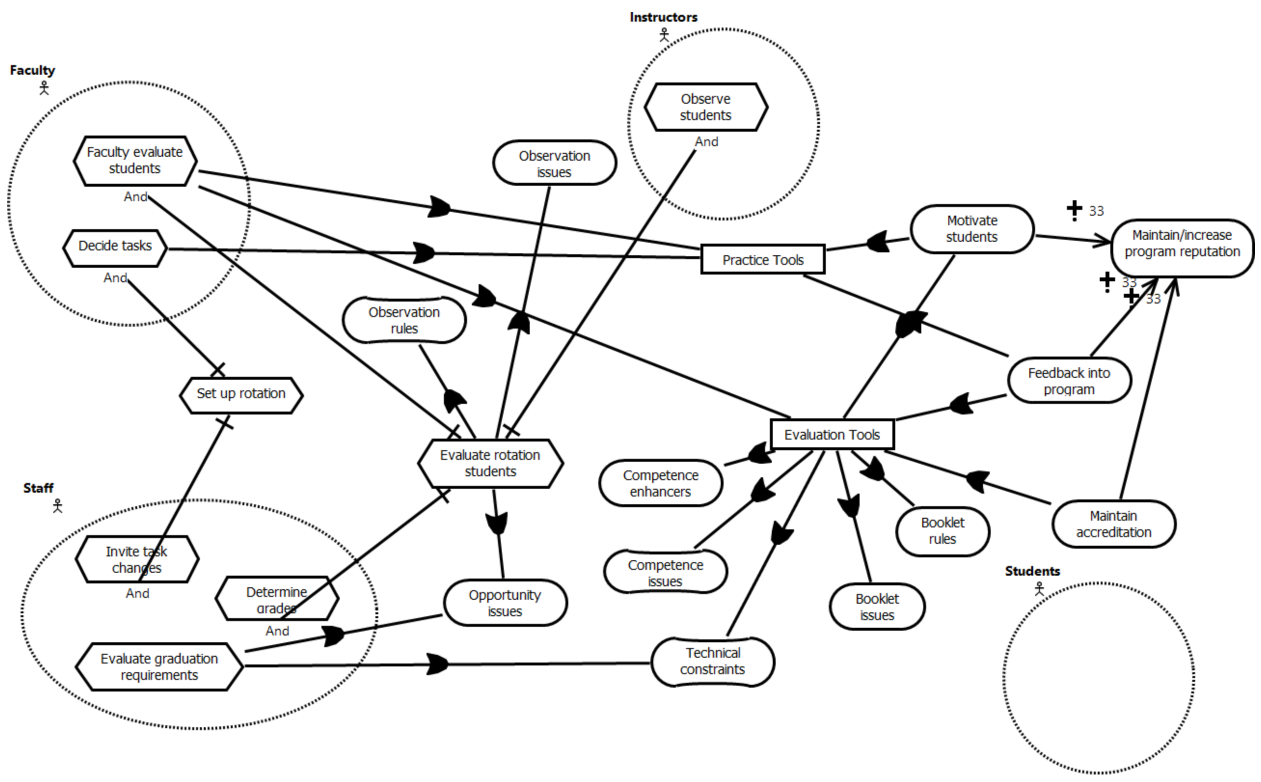


Figure 21. Goal model with dependency information added.

This example is still in progress. In particular, no DoLs associated with Students have been added to the AT model, and thus no Student Actor Tasks are present in the Goal model.

7. Issues and Limitations

The problems encountered while developing this tool fall into four categories: development by experimentation, platform constraints, little testing, and no user studies. Each of these issues is discussed in more detail in the following sections. A short discussion regarding building this tool, installing it, and using it is presented at the end of this section.

7.1 Experimentation

Most of the development of the VS AT tool took the form of experimentation to figure how to accomplish something. There is a good set of tutorials that cover many of the things a DSL developer might want to achieve, but several things we needed to add to the language and tool were not covered. An example is the ability to identify elements whose names should be displayed in the compartments of an ASD compartment shape. Also, given the great flexibility of the DSL SDK there are no doubt more efficient ways to achieve the functionality we desired in the VS AT tool.

Beyond these DSL tool questions, a much larger experiment discussed in Section 7.1.2 (Natural Language Processing Experiments) involved using Apache NLP to attempt to automatically find contradictions in AT models.

7.1.1 Visualization

When the Vector Surveillance example was specified using the VS AT tool, it became abundantly evident that visualization of any but the simplest AT models is an issue. To help ameliorate this problem several user commands were added. Most have to do with hiding or showing specific relations in the model. For example, hiding all relations from an ASD to its related ASDElements and Mediations is almost always a good idea. User commands were also added to hide or show the other relations within an ASD that relate ASDElements to each other (e.g. whoDoesDoL), and to hide or show relations between ASDs (network and hierarchical). Mediation objects are annotated with the mediating element and the two mediated elements, so no relations need be shown for these objects. Also, the zoom commands were added so that the modeler can have an overall view of the model.

7.1.2 Natural Language Processing Experiments

Contradiction analysis is a rich area for exploring how activity systems might evolve over time and the subsequent influence of various evolutions on the system. Engeström identified four types of contradictions, all of which deal with the semantics of an ASD and its constituent ASDElements. Since each element has a description which typically consists of natural language, automated discovery of contradictions is a challenge. This section describes some experiments that were performed to see whether the Apache OpenNLP library could be integrated into the VS AT tool to identify potential contradictions based on syntactic analysis of ASDElement descriptions.

A command was added to the user command menu, along with C# code to use the Apache OpenNLP library. These java libraries had to be converted to be used by C# methods, and this was accomplished using ikvm, and is discussed in the next sub-section.

Regular expressions were used to identify sentence nouns and verbs in the element descriptions but this was not successful since descriptions are rarely in the form of simple sentences. However, we demonstrated that it is possible to identify duplicate words within Rules, DoLs, and Outcomes, and between Rules and DoLs and DoLs and Outcomes provided the descriptions are re-written as sentences. The rationale for this approach is that within an element type such as Rules, if the subject (in the English grammar sense) is the same in multiple Rules, then there might be a contradiction between the rules. Similarly, if the object (in the English grammar sense) is the same in multiple rules, a contradiction might exist. For our initial experiments we simplified the problem to one of searching for common nouns and verbs in both parts of the descriptions.

We were able to successfully find duplicate words, however this ability is too limited to be of general use in identifying contradictions. WE therefore decided to add completely user-defined contradictions at this time. We created a new AT language domain class, Contradiction, which the modeler can use to relate any elements that are contradictory. We initially defined this class as having relations to any number of ASDs or ASDElements, however this assumes that all items related in this way contradict each other, which is not generally true. Consider the following Rules:

- R1: P decides where to go and what to do
- R2: P can delegate decisions to others
- R3: G decides what to do

Here R2 and R3 probably contradict R1 – if P must make the decision, then it cannot be delegated, and G cannot decide what to do. However, R2 and R3 do not contradict each other. Thus, we conclude that contradiction does not support a transitive property – elements may only be contradictory in a particular context. We therefore added a specific relation to the item that identifies the one element (or ASD) that provides the context where the other elements (or ASDs) are contradictory. We call this the considered item. The other elements (or ASDs) that contradict it are referenced using a different relation.

Details – Creating the DLL for Open NLP

Several tools and libraries had to be downloaded for this task:

- ikvm
- OpenNLP
- UIMA SDK
- Javamail

The jar files were converted to .Net libraries (.dll files) using the ikvm tool. The .dll libraries have to be 'strongly named' for use in Visual Studio. This means that the namespaces are unique and this is accomplished by first creating a key file from a Visual Studio tool called *sn*, and then using an ikvm option when creating the .Net libraries. The resulting libraries must be added as references to the Visual Studio project, in the DslPackage solution. To do this, from the References item right hand click and select Add Reference, then select the Browse button, browse to where the libraries are, and choose each library and finally click Add. Once this is done the libraries can be added to custom code as references. For example, the following code shows the initialization of some of the features we used in our experiments:

```
using opennlp;
using opennlp.tools;
using opennlp.tools.parser;
using opennlp.tools.tokenize;
using java.io;

…

// Set up the tokenizer and parser by opening the models and loading them. The
//parser especially takes a long
// time, so this should be done only once. It seems to be the parser factory that
//takes all the time.
java.io.FileInputStream parIstr = new
       java.io.FileInputStream("c:\\opennlpModels\\en-parser-chunking.bin");
java.io.FileInputStream tokIstr = new
       java.io.FileInputStream("c:\\opennlpModels\\en-token.bin");
opennlp.tools.tokenize.TokenizerModel tokModel = new
       opennlp.tools.tokenize.TokenizerModel(tokIstr);
opennlp.tools.tokenize.TokenizerME tokenizerToUse = new
       opennlp.tools.tokenize.TokenizerME(tokModel);
ParserModel model = new ParserModel(parIstr);
Parser parserToUse = ParserFactory.create(model);

…
```

Details – Using OpenNLP to Analyze ASDElement Descriptions

A user command was written to find duplicate words within Rules, duplicate words within DoLs, and duplicate words within Outcomes, and then duplicate words between Rules and DoLs, and between DoLs and Outcomes. The rationale for these choices is that the descriptions of Rules, DoLs, and Outcomes tend to be quite a bit longer than those of Tools, Subjects, Aims, or Community members, although Aims can also have long descriptions. It seems that duplicate nouns or verbs in these descriptions could indicate contradictions. For example, if there are multiple rules associated with the same item (a noun) or action (a verb) there might be contradictions. The first step is to create tokens of the description string and then to parse it. Next regular expressions are used to find the noun part of the subject portion of the description, the verb part of the description, and any nouns in the verb part. The results of matches that occur are put into HashSets. Finally the hash sets are compared to find duplicates.

We immediately ran into issues because the descriptions were usually not simple sentences, so several had to be re-written in order for the parsing and regular expression matching to work. Next we found that since words may have similar meanings but not be the same, some duplications were missed. Finally, the

OpenNLP tools sometimes confuse nouns and verbs. For example the word ‘check’ can be either a noun or a verb, but the library seems to assume it is noun. This problem also led to changing the descriptions so that they would work with the parser.

An excerpt of the command output for the Take a Vacation ASD example is given below:

```
ASD TakeVacation: Rules:
	1. Agent provides feedback about possible issues
	2. Planner delegates arrangements or decisions to others in the group
	3. Planner makes decisions on where to go

ASD TakeVacation: Rule duplicate words:
	Planner: 2 3
	decisions: 2 3

ASD TakeVacation: DoLs:
	1. Planner finds out options
	2. Planner decides where to go
	3. Planner buys tickets
	4. Planner makes personal arrangements
	5. TravelGroup makes personal arrangements
	6. TravelGroup goes to check in
	7. Agent makes bookings
	8. Agent provides information
	9. ActivityCompany provides opportunities
	10. ActivityCompany creates advertising
	11. ActivityCompany provides information

ASD TakeVacation: DoL duplicate words:
	Planner: 1 2 3 4
	makes: 4 5 7
	arrangements: 4 5
	TravelGroup: 5 6
	Agent: 7 8
	provides: 8 9 11
	information: 8 11
	ActivityCompany: 9 10 11

ASD TakeVacation: Duplicate words in Rules (L1) and DoLs (L2):
	Agent: L1-1 L2-7 L2-8
	provides: L1-1 L2-8 L2-9 L2-11
	Planner: L1-2 L2-1 L2-2 L2-3 L2-4 L1-3
	arrangements: L1-2 L2-4 L2-5
	makes: L1-3 L2-4 L2-5 L2-7
```

The last section is interpreted as follows:

- the noun ‘Agent’ appears in Rule 1 and DoLs 7 and 8
- the verb ‘provides’ appears in Rule 1 and DoLs 8, 9, and 11
- the noun ‘Planner’ appears in Rules 1 and 3and DoLs 1-4
- the noun ‘arrangements’ appears in Rule 2 and DoLs 4-5
- the verb ‘makes’ appears in Rule 3 and DoLs 4, 5, and 7

To demonstrate the steps, consider the first Rule. The result of tokenizing and parsing this description is:

```
(TOP (S (NP (NNP Agent)) (VP (VBZ provides) (NP (NP (NN feedback)) (PP (IN
about) (NP (JJ possible) (NNS issues)))))))
```

We separate the sentence into a subject part:

```
(NP (NNP Agent))
```

and a verb part:

```
(VP (VBZ provides) (NP (NP (NN feedback)) (PP (IN about) (NP (JJ possible) (NNS
issues)))))
```

Next we look for noun words in the subject part and get: `Agent`, which we add to a hash set. We then look for verbs in the verb part and get `(VBZ provides)` from which we extract `provides`, and we add that to the hash set. Finally we look for nouns in the verb part and extract the nouns which give `feedback` and then `issues`, each of which we add to the hash set. We create a list of hash sets, one for each rule. The hash set for the first rule is as follows:

```
[0] “Agent”
[1] “provides”
[2] “feedback”
[3] “issues”
```

The hash set for the second rule is as follows:

```
[0] “Planner”
[1] “delegates”
[2] “arrangements”
[3] “decisions”
[4] “others”
[5] “group”
```

The hash set for the third rule is as follows:

```
[0] “Planner”
[1] “makes”
[2] “decisions”
```

Once all the Rules have been processed we have a list of hash sets. We look for non-null intersections in the contents of these sets, taken two at a time. Each time there is a non-null intersection we can determine which word of which Rules is duplicated and write these to a report. For example, the word “`Planner`” appears in both Rules 2 and 3, and the word “`decisions`” also appears in both Rules 2 and 3.

The main shortcomings of this approach include the fact that the descriptions have to be in a specific format (i.e. simple, complete sentences), and that contradiction analysis really needs to have a flexible implementation of a group of similar words or ideas so that duplicates can be discovered. Of course, duplicates are only one possible type of contradiction indicator. Phrases or ideas that are opposites might also indicate contradictions. Finally, relations should be used to find potential contradictions – for example, any DoL with multiple relations to Community members (whoDoesDoL) may include contradictions depending on how the task is partitioned, what rules are related to it (rules2dols) and what tools are used for it (tools2dols).

7.1.3 GRL Mappings

The tracelinks we have identified between ASD elements and GRL goal model elements are used to generate a text file indicating GRL elements that should be created to transform the ASD model into a goal model. Some of the ASD relations can be used to further augment the goal model, and these have been included in the C# code that analyzes the ASD. Many issues were discovered when creating these mappings, as were discussed in the proposed electronic task booklet example above. Some AT relations supported by the VS tool are not yet incorporated into these mappings, specifically Outcome network relations and hierarchical decomposition relations.

Perhaps more significant, ASDElement refinement has not been added to the VS metamodel and therefore it is not available in the AT language tool at this time, even though it was defined in the first AT language metamodel we created. This relation can be defined between elements of the same type, and if it was available in the VS tool then information obtained from it could be used to preserve original information in situations where the model is too complex and abstraction needs to be applied. Substantial work needs to be done however, to develop a method to visualize refined elements. Since this relation is not available, all detailed information related to abstractions such as those discussed in the electronic task booklet example must be documented outside of the tool.

An additional issue is that there is no metamodel relation between particular Aims and Outcomes, which would be useful when there are both multiple Aims and Outcomes, and the positive contributions of the Aim-mapped Goals are being created to the Outcome-mapped Goals.

A final issue is that of improving the decision of whether a Task or Resource depends on a specific Rule-mapped Goal/Softgoal. One idea is to extend the AT metamodel to include a concept that encompasses this trinary relation. That is, a Rule is related to a DoL and potentially also to a Tool used to accomplish the DoL. If there is no assocated Tool, then the Task mapped from the DoL is the item dependent on the Rule-mapped Goal/Softgoal. Otherwise, the Tool-mapped Resource may be dependent on the Rule-mapped Goal/Softgoal.

7.1.4 ASDElements Related to Multiple ASDs

The AT metamodel allows ASDElements to be related to more than one ASD. This reduces redundancy in the set of model elements. However, it presents a problem when faced with reasoning about relations between elements that are ASD context-specific. This issue was mentioned in the section on Validation Custom Code, where the intra-ASD relations were checked to make sure that the source and target were related to the same ASD. The workaround adopted in the examples is that only ASD Community elements are related to multiple ASDs. Further experimentation and research is needed to resolve this issue.

7.2 Platform Constraints

The AT language tool is implemented for Visual Studio only. We expect that many of the techniques will apply to subsequent tool versions, but since there is so much custom code involved it is not clear how much will need to be actively maintained for newer versions of Visual Studio.

7.3 Testing

Only the most basic testing has been performed on the tool, although the code has been studied for common errors. Clearly the different example systems that were input to the tool also provided testing, but testing has not been systematic.

7.4 User Studies

No user studies have been performed, and only the developer has used the tool. Input from an HCI expert was solicited and used for some display issues.

7.5 Tool Access – Building, and Installing the Tool, and Creating AT Models

To build the AT Language tool, the Build Solution button on the BUILD menu can be used. Then it is necessary to install it from VSIX file. This file can be found in the Projects folder in the specific project folder. For example:

Project/ActivityTHeoryV5/DslPackage/bin/Debug/CSU.ActivityTheoryV5.DslPackage.vsix

This file can be executed by clicking on it, and choosing the installation button on the dialog box that is presented. When Visual Studio is started, from the **TOOLS** command, there is a button called '**Extensions**

and Updates…’. Selecting this button brings up a window showing all of the extensions installed, one of which should be ActivtyTheoryV5.

To create a project that uses AT models, use the **FILE** command, the **New** button, and then **Project**. Select a **Visual C# template**, and a **WPF Application**. Choose a name for the project, then press the OK button. From the **Solution Explorer** tab, right click on the name of the project, select **Add**, then **New Item**. Scroll down the choices until ActivityTheoryV5 appears and select it. Choose a name for the model making sure that the file extension remains ‘**.atv5**’, then select **Add**. A new model diagram will be displayed, with the AT editor on the left side. The only thing that is initially added to the diagram is the color key, which usually needs to be resized and moved.

8. Conclusions

This report has described a Visual Studio tool that realizes an Activity Theory domain specific language. Our conclusions regarding this work fall into four areas. The first relates to using the DSL Modeling SDK to create the tool. The SDK is quite complex in its structure of generated projects and where custom code must be added to implement desired functionality. While the tutorials regarding the SDK are comprehensive, they have not been updated to work with newer versions of VS so some details do not work and must be determined from other sources. Other details are not included (e.g. how to filter a list for display in a graphical editor shape compartment) and must be found from user groups and the web in general. However, we found that all of the functionality we wanted to include was possible one way or another using custom code. User commands were especially useful to try different additions – this method is very flexible with the caveat that it is easy to have too many commands. Finally, the ability to add custom code to relation builders is a good way to help modelers; relations that are not allowed cannot be created.

A second area of conclusions relates to this tool and the previous USE realization of the AT language. This tool was much more difficult to implement than the USE model. However, it was possible to experiment with functionality such as GRL mapping and NLP analysis, which would be very difficult if possible at all in USE. We were also able to add functionality to this tool to make object creation much easier than is possible in the USE tool, especially with the user command to create a set of objects given a type and a list of descriptions. However, the modeler must still add the relations between the newly created objects and the related ASD, and between the objects manually. Note that in the USE tool, the user can specify these relations in a text file that is read in to create/modify an object model.

Our third set of conclusions relates to using the VS tool in our on-going research. On the one hand, once the structure of the DSL Modeling SDK solutions is understood, changes are not very difficult to try experiments or to add new functionality. On the other hand, fundamental changes to the language metamodel can cause some or all of the custom code to break, and render models created with prior versions unusable. We encountered this problem when we replaced concrete classes for each of the seven types of ASD elements with the two-level abstract class hierarchy of ASDElement and MediaTINGEle/MediaTEDEle between the concrete classes and the ATModelRoot class. In the end we simply created an entirely new tool based on this new metamodel rather than attempt to migrate all the custom code from previous versions.

Our fourth area of conclusions deals with using the VS AT tool for something other than research. Clearly systematic functional testing and user testing would be required, and changes made to the tool would be required. Currently the only way to create new AT models is by creating a new project that is a C# WFP application and adding new AT model items to the application. It is expected that the ability to create a new AT model project directly would be needed. Finally, this tool can only be used from Visual Studio, and it is not clear how well it will migrate to newer versions of VS. It is possible that solutions exist for these issues, but they need to be fully investigated and solved.

With respect to our original motivations for developing this tool we note the following. First, we were very successful in using the DSL SDK to develop the AT tool and associated graphical editor. We also were successful in creating custom code to implement both structural and other constraints. We were able to explore and extend our mapping to GRL Goal model elements based on AT models. However, while object model visualization experiments are easier than with the USE tool (e.g. hiding both objects and relations with user commands), it is not clear that object model creation is much easier with the VS AT tool. Using a graphical editor, clicking a mouse to select a relation type, and clicking on the objects at the ends of the relation is easy in the VS AT tool, but creating many relations is still cumbersome. We expect that experiments to address these issues using the VS DSL SDK will be very feasible.

References

[1] L. Eisen, J.M. Bieman, S. Ghosh, and S. Lozano-Fuentes, "Using cell phones for entry to a Dengue Decision Support System", Research Project supported by Award Number R21AI080567 of the National Institute of Allergy and Infectious Diseases, 2009-2012; http://www.cs.colostate.edu/ddss/ (acc. Jan. 2013)

[2] Y. Engeström, Learning by expanding, Helsinki: Orienta-Konsultit, 1987.

[3] Geri Georg & Lucy Troup, "Experiences Developing a Requirements Language Based on the Psychological Framework Activity Theory", Proceedings of the MODELS 2013 OCL Workshop, CEUR Workshop Proceedings, Vol 1092, http://ceur-ws.org/Vol-1092, ISSN 1613-0073, pp 63-72, 2013.

[4] Geri Georg & Gunter Mussbacher, "USE Tool Analysis of Activity Theory Models", Colorado State University Technical Report, CS-13-102, March, 2013

[5] Geri Georg & Robert France, "An Activity Theory Language: USE Implementation", Colorado State University, Computer Science Technical Report, CS-13-101, January, 2013.

[6] ITU-T, *User Requirements Notation (URN) – Language definition*, ITU-T Recommendation Z.151 (10/12), Geneva, Switzerland, October 2012; http://www.itu.int/rec/T-REC-Z.151/en (acc. March 2013).

[7] jUCMNav website, http://softwareengineering.ca/jucmnav (accessed March 2013).

[8] S. Lozano-Fuentes, S. Ghosh, J.M. Bieman, D. Sadhu, L. Eisen, F. Wedyan, E. Hernandez-Garcia, J. Garcia-Rejon, and D. Tep-Chel, "Using Cell Phones for Mosquito Vector Surveillance and Control", *24th International Conference on Software Engineering & Knowledge Engineering (SEKE'12)*, Knowledge Systems Institute Graduate School, pp. 763–767 (2012).

[9] R.F. Paige, N. Drivalos, D.S. Kolovos, K.J. Fernandes, C. Power, G. K. Olsen, and S. Zschaler, "Rigorous identification and encoding of trace-links in model-driven engineering", *Software & Systems Modeling*, vol. 10, no. 4, pp. 469–487 (2011).

Appendix A – Activity Theory Background

Activity Theory is a psychological framework that can be used to analyze human activity. It was developed in the 1920's and 1930's by Vygotsky and introduced the notion that an individual's actions are mediated by various elements. Later Leont'ev expanded these ideas to include the idea that human activity is never isolated, but instead takes place in some context that is influenced by societal concerns. Leont'ev moved the focus of activity analysis from the individual to the community. Most of the work extending these ideas and adapting them for use in the context of learning and later, cooperative work, was done by

Engeström. He developed the theory to include learning that takes place over time and that drives evolution of human activity in response to contradictions in the activity system. A major tenant of Leont'ev and Engeström's work is that human activity must be analyzed holistically as a system; individual elements of an activity system cannot be analyzed in isolation. Further, human activity systems do not usually occur in isolation, but rather as a set of networked activities, where the outcome of some activity may serve as one of the other elements (e.g. tool) in another. Engeström introduced a diagram of an activity system. This is shown in Figure A1.

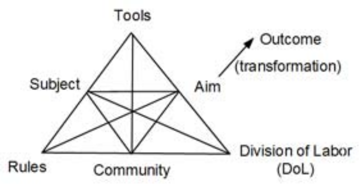


Figure A1. Engeström Activity System Diagram (ASD)

The diagram consists of two triangles with joining lines between their vertices. The vertices of the outermost triangle are mediaTING elements of the activity system, while the vertices of the inner triangle are mediaTED elements of the system. The diagram is interpreted as follows. First, every activity system has some aim(s); the reason why the activity is being undertaken. As a result of performing the activity, some transformation occurs to produce the outcome(s) of the activity. The subject(s) use mediating tool(s) to achieve the aim(s) of the activity. Subjects are part of a community, and the interactions between subjects and members of the community are mediated by rules. Rules are the norms and conventions associated with these interactions. Finally, the community consists of everyone with an interest in the aim(s), and the division of labor is how the community divides the work that has to be done to achieve the aim(s).

From this interpretation, we see that the prime mediations are that Tools mediate between Subjects and Aims, Rules mediate between Subjects and Community members, and Division of Labor items (DoLs) mediate between Community members and Aims. The additional lines in the diagram indicate other, secondary, possible mediations. For example, the line between Community and Tools indicates that it is possible for a Tool to mediate between a Community member and an Aim or between a Community member and a Subject. Similarly, a Rule could mediate between a Community member and an Aim, or between a Subject and Aim. A DoL could mediate between a Community member and Subject, or between a Subject and Aim. Although any of these mediation relations is possible, we have found that it is fairly straightforward to initially analyze only the primary mediations in our work, and these are the mediations that we have chosen to support in our Visual Studio AT Language tool analysis user command.

Engeström discusses the idea of contradiction as a driver of evolution of activity systems. He defines four types of contradictions. The first type of contradiction can occur within a set of elements of the same type in an ASD. For example, two Rules in one ASD may be contradictory. The second type occurs across elements in an ASD. An example of this is if a particular DoL contradicts a Rule. The third type of contradiction occurs between different evolutions of the activity, where the activity is still in the process of transitioning to the more advanced version. An example is when an activity has a tool that is used in an ad-hoc manner and additional constraints are put on its use in order to bring more control and predictability to its use. The activity may be evolving into a more predictable activity by using the tool in a constrained way, but other elements in the activity system (e.g. the subjects using the tool) may find the controls onerous or prohibitive towards achieving the activity aims. The fourth type of contradiction occurs across multiple ASDs in a network. A common situation is that an Outcome from one activity may serve as a Tool in another. An example of such a contradiction occurs when the outcome of one activity is research software, which in turn is used as a production tool in another activity.

In all of these cases the activity system evolves to alleviate the contradictions, in fact the contradictions are necessary precursors to activity evolution.

Leont'ev, Alexei N., Activity, Consciousness, and Personality, Englewood Cliffs, NJ: Prentice-Hall, 1978.

Vygotsky, Lev, Mind in Society: the development of higher psychological processes, Cambridge: Harvard University Press, 1934.

Appendix B – Trace-Links between AT and GRL

We use trace-links to define mappings between AT language elements and GRL elements. Trace-links enable bi-directional mappings, which are important in our on-going work, to map between augmented and modified Goal models and ASD networks. (Note that the tool described in this report does not contain features to automate creating GRL elements, only textual suggestions for how a related Goal model can be manually created.) In addition to the graphical specification shown in Figure B1, trace links allow us to specify additional constraints on the mappings. These constraints can be specified in the Epsilon Validation Language. For simplicity we show them in English.

The trace-links shown in Figure B1 are informally defined as follows:

- AT Community objects are mapped to GRL Actors. Since the AT metamodel requires that each Subject be related to one community member (subjComm relation), it is not necessary to map AT Subject objects to any GRL elements.
- AT DoL objects are mapped to GRL Tasks.
- AT Tool objects are mapped to GRL Resources.
- AT Outcome and Aim objects are mapped to GRL Goals. (Goals have clear conditions for being met, whereas Softgoals do not. In an ASD, it is necessary to know when Aims and Outcomes have been achieved, or else the activity may never end, so it is reasonable to map these AT elements to Goals.)
- AT Rules are mapped to either Goals or Softgoals – in many cases the Rule may be more of a guideline and it may not be clear if it is met or not, in which case it maps better to a Softgoal than a Goal.

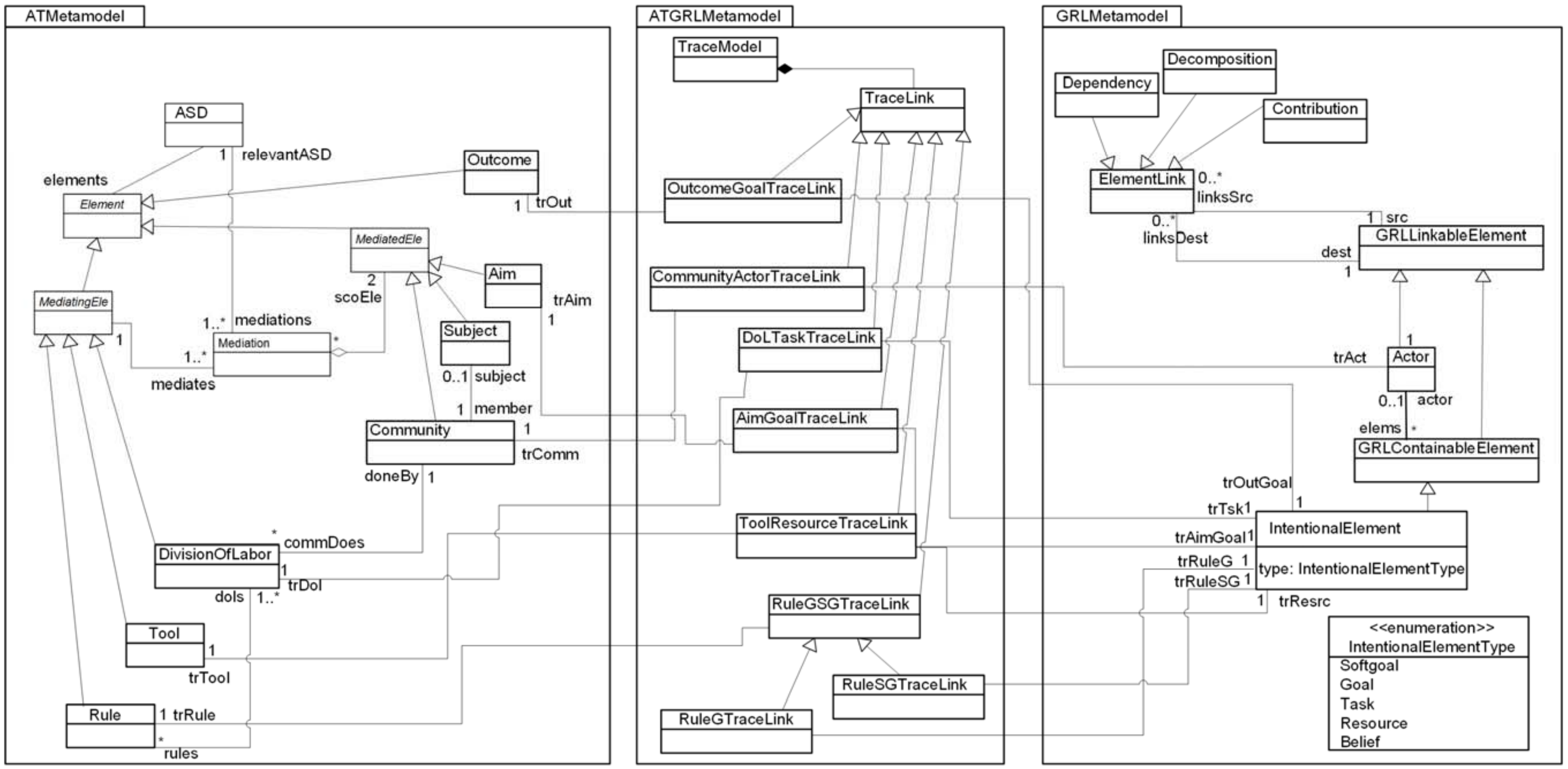


Figure B1. Trace-links between AT metamodel elements and GRL metamodel elements

Other additions can be made to the Goal model depending on relations that exist in the ASD:

- **whoDoesDoL** relation: If there is a single Community object related to a DoL object by this relation, then the respective GRL Task can be placed inside the respective GRL Actor sphere (using the GRL actor/elems relation shown in the GRL metamodel portion of Figure B1). If there are multiple Community objects related to a DoL by this relation, then the respective GRL Task needs to be decomposed (using an AND decomposition) into sub-Tasks that are accomplished by each of the respective Actors, and be placed in these Actors' spheres. The super-Task should not have any actor/elems relation, that is, it should be placed outside of all Actors' spheres.
- **rules2dols** relation: If there is a single DoL object related to a Rule object, then a dependency relation can be made between the respective GRL Task and the respective Goal or Softgoal (the Task depends on the Goal/Softgoal). If there are multiple DoLs related to this Rule, then each respective task is dependent on the respective Goal/Softgoal. Additional considerations occur if the DoL is also related to some Tool(s) via the tools2dols relation since the Rule may actually apply to the Tool being using in the context of the DoL.
- **tools2dols** relation: If there is a single DoL object related to a Tool object, then a dependency relation can be made between the respective GRL Task and the respective Resource. If there are multiple DoLs related to this Tool, then each respective task is dependent on the respective Resource.
- **Tool-Subject/Aim** mediations: Dependencies can be created as follows. First, recall that the Tool object will be mapped to a GRL Resource, the Subject object (using the subjComm relation to a Community object) is essentially mapped to a GRL Actor, and the Aim object is mapped to a GRL Goal. The mediation means that the respective Goal is dependent on the respective Resource and the respective Actor is also dependent on that same Resource. However, an Actor cannot be dependent on a Resource – some Goal/Subgoal or Task must be dependent on the Resource. Therefore, the tools2dols relation can be used to find potential Tasks which might be dependent on the Resource. However, any DoL objects identified through this relation must have the Community object related to the Subject object in their set of whoDoesDoL Community objects. If such a DoL is found, then the related Task can be specified as being dependent on the resource.
- **GRL Goals mapped from AT Aims**: All such Goals can be defined as having positive contribution links to the GRL Goals mapped from AT Outcomes of the ASD. Note that in the case of multiple Aims and Outcomes, there is currently no way to determine which Goal mapped from an Aim contributes positively to which Goal mapped from an Outcome.

Appendix C – Abstraction Information for the Proposed Electronic Task Booklet

Each of the original ASD elements for this example were categorized into abstracted elements described in the following tables. The abstracted element is listed in the first column, with the original elements that were grouped and abstracted to create it in the second column. The third column lists any mediations related to the element in the case of a mediating element (i.e. Rule, Tool, or DoL). The last column lists any additional relations that the element has to other elements in the ASD. Note that the Mediation column is not included in the tables for elements that are not mediating elements. In the case of DoLs, those with mapped GRL Tasks that need to be decomposed have a fifth column with this information.

Table C1. Abstracted Rules

Abstracted Rule	**Specific Rules**	**Mediations**	**rules2dols**
Observation Rules	The honor system is critical Professional organization wants direct observation and signing off on task within a short time (<1day?) Senior year is for working in real situations, previous 3 years are for lectures/labs Senior rotations have a maximum of 8 students so direct observation is possible	SC: This rule mediates between Subject Faculty, and Community member Students SC: This rule mediates between Subject Students, and Community member Instructors	This rule is related to DoL 'Evaluate rotation students'
Observation Issues	Junior labs have more students, so direct observation is not always possible Physical issues – some areas require environments that do not allow electronic or even paper booklets Often it isn't feasible/possible for rotation faculty to directly observe and sign off on a task A junior in a rotation may demonstrate a skill but a senior may document it Rotation faculty may not all supervise all weeks of the rotation Sometimes rotation faculty may have to take over from students because of time constraints	SC: This rule mediates between Subject Faculty, and Community member Students SC: This rule mediates between Subject Students, and Community member Instructors	This rule is related to DoL 'Evaluate rotation students'
Booklet Rules	Booklets are pretty self-explanatory Seniors have to have senior task booklet complete to graduate There is only one rotation that requires the related skills in the task booklet to be complete in order to pass the rotation Professional organization requires Junior and Senior data as part of the accreditation process	SC: This rule mediates between Subject Faculty, and Community member Students SC: This rule mediates between Subject Students, and Community member Instructors SC: This rule mediates between Subject Students, and Community member Staff	This rule is related to DoL 'Evaluate rotation students'
Booklet Issues	Faculty need to be reminded of the importance of the task booklets and direct observations Students have to be reminded to get signed off on tasks in booklets Students don't always have booklets on their persons when they do/are observed doing a task Staff: students have the responsibility to know what tasks need evaluation/signing off Students: it would be nice if the faculty knew what tasks will be available during a rotation Some tasks may be hard to complete because there are few opportunities	SC: This rule mediates between Subject Faculty, and Community member Students SC: This rule mediates between Subject Students, and Community member Instructors	This rule is related to DoL 'Evaluate rotation students'
Competence Enhancers	The rotation evaluation program form has questions based on competence areas defined by professional organization	SC: This rule mediates between Subject Faculty, and Community member Students	This rule is related to DoL 'Evaluate rotation students'

Competence Issues	If you are going to evaluate competence in a task, a sliding scale (0-5) would be better than meets/exceeds expectations If you are going to gather evaluation data, then use it for something Rotation faculty can decline to do an evaluation if they didn't have a student for one of the rotation weeks	SC: This rule mediates between Subject Faculty, and Community member Students SC: This rule mediates between Subject Students, and Community member Instructors	This rule is related to DoL 'Evaluate rotation students'
Opportunity Issues	Separate organization sets number of rotations and students based on the number of different kinds of needs they are predicted to have Separate organization gets tertiary referrals for most cases Booklets don't record the number of times a student did a task, so they can't yield data that would help modulate the provided opportunities across labs and rotations	SC: This rule mediates between Subject Faculty, and Community member Students	This rule is related to DoL 'Evaluate rotation students' This rule is related to DoL 'Evaluate graduation requirements'
Technical constraints	Network bandwidth is limited Rotation evaluation program can't be used for task booklet There should be no extra work required for 'booklets', which implies a single recording media; gathering data from multiple sources is extra work	SC: This rule mediates between Subject Faculty, and Community member Staff	This rule is related to DoL 'Evaluate rotation students' This rule is related to DoL 'Evaluate graduation requirements'

Table C2. Abstracted DoLs

Abstracted DoL	**Specific DoLs**	**Mediations**	**whoDoesDoL**	**Decomposed GRL Tasks**
Rotation setup	Practicum Coordinator invites rotation faculty to change task lists once a year Rotation faculty decide tasks Student coordinator gathers task lists and prints booklets Rotation Faculty can change rotation evaluation program forms once/year	CA: This DoL mediates between Community member Faculty, and Aim Motivation CA: This DoL mediates between Community member Faculty, and Aim Accreditation	This DoL is related to Community member 'Faculty' This DoL is related to Community member 'Staff'	Faculty decide tasks Staff invite task changes
Rotation evaluation	Rotation Faculty must evaluate students for their grade Dean's office uses rotation evaluation data to grade student rotations and determine class standings Rotation faculty, faculty, post-professional degree graduates, and technicians sign off on booklet tasks	CA: This DoL mediates between Community member Faculty, and Aim Motivation CA: This DoL mediates between Community member Faculty, and Aim Accreditation CA: This DoL mediates between Community member Instructors, and Aim Motivation CA: This DoL mediates between Community member Faculty, and Aim Feedback into Program	This DoL is related to Community member 'Faculty' This DoL is related to Community member 'Instructors' This DoL is related to Community member 'Staff'	Faculty evaluate students Instructors observe students Staff determine grades
Graduation requirements evaluation	Student Coordinator determines if booklets complete for graduation	CA: This DoL mediates between Community member Staff, and Aim Accreditation	This DoL is related to Community member 'Staff'	N/A

Table C3. Abstracted Community Members

Abstracted Community member	**Specific Community members**
Faculty	Rotation Faculty General Faculty
Staff	Staff Admin Student Coordinator Practicum Coordinator
Instructors	Post-Professional Degree Graduates Technicians
Students	Juniors Seniors

Table C4. Abstracted Subjects

Abstracted Subject	**Specific Subjects**	**subComm**
Faculty	Rotation Faculty	This Subject is related to Community member 'Faculty'
Student	Student	This Subject is related to Community member 'Students'

Table C5. Abstracted Tools

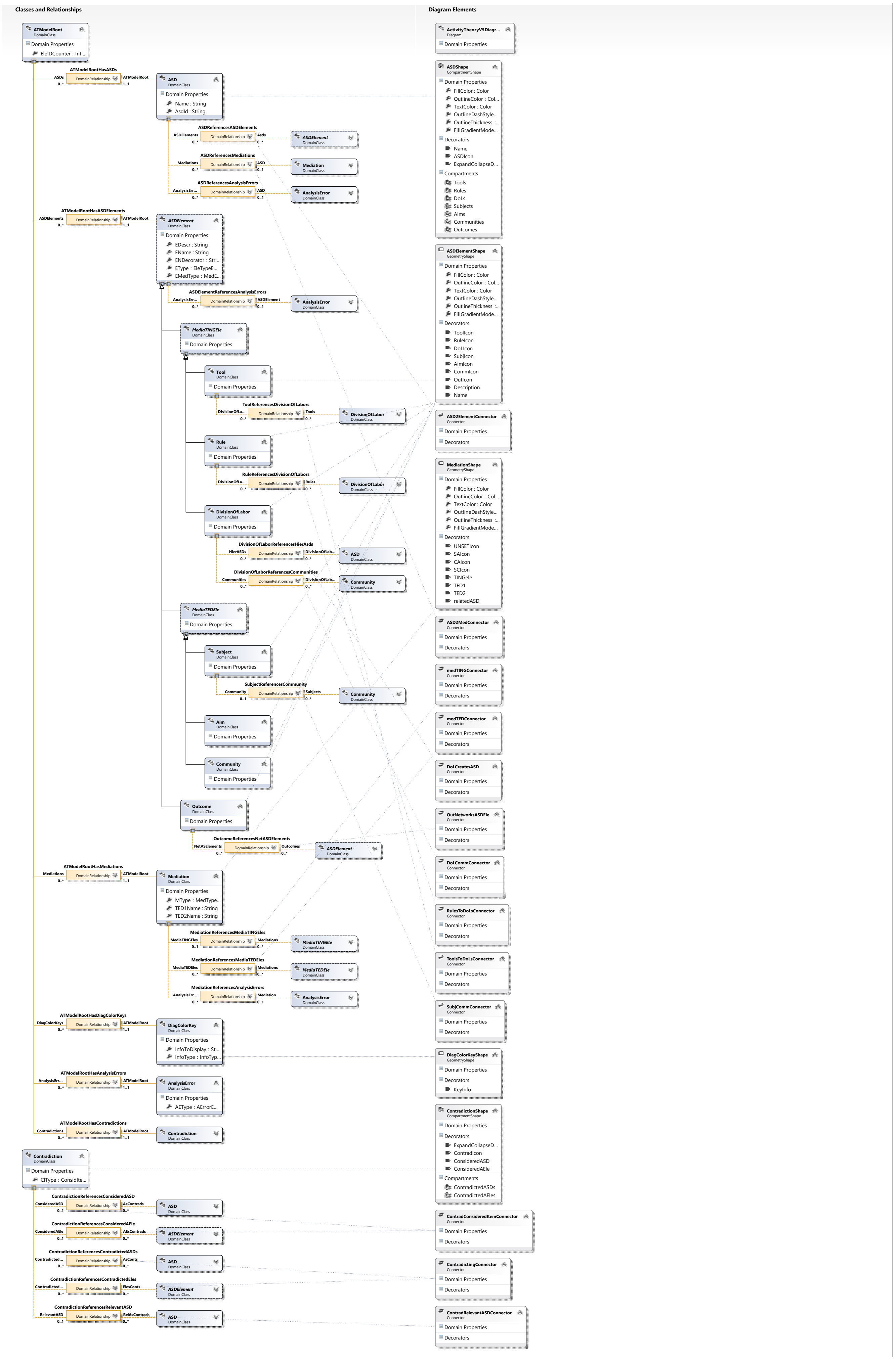
Abstracted Tool	**Specific Tools**	**Mediations**	**tools2dols**
Evaluation Tools	Junior & Senior Task Booklets Rotation evaluation program	SA: This DoL mediates between Subject Faculty, and Aim Motivation SA: This DoL mediates between Subject Faculty, and Aim Accreditation SA: This DoL mediates between Subject Faculty, and Aim Feedback	This Tool is related to DoL 'Evaluate rotation students' This Tool is related to DoL 'Evaluate graduation requirements'
Practice Tools	Rotations Teaching Labs Other computer programs	SA: This DoL mediates between Subject Faculty, and Aim Motivation	This Tool is related to DoL 'Set up rotations' This Tool is related to DoL 'Evaluate rotation students' This Tool is related to DoL 'Evaluate graduation requirements'

Table C6. Abstracted Aims

Abstracted Aim	**Specific Aims**
Motivate students	Demonstrate task opportunities Provide data to motivate students to increase competencies
Maintain accreditation	Record tasks performed Provide evidence of task competence to professional organization
Feedback into program	Record level of task competence Provide data to align task opportunities in classes and rotations

Table C7. Abstracted Outcomes

Abstracted Outcome	**Specific Outcomes**
Maintain/Increase program reputation	Students walking out the door with a professional degree are able to perform modern, routine complex tasks property without supervision Competence data available for professional organization accreditations



Classes and Relationships
Diagram Elements
ATModelRoot
DomainClass
Domain Properties
EleIDCounter : Int...
ATModelRootHasASDs
ASDs
0..*
DomainRelationship
ATModelRoot
1..1
ASD
Name : String
AsdId : String
ASDReferencesASDElements
ASDElements
Asds
ASDElement
ASDReferencesMediations
Mediations
Mediation
0..1
ASDReferencesAnalysisErrors
AnalysisErr...
AnalysisError
ATModelRootHasASDElements
EDescr : String
EName : String
ENDecorator : Stri...
EType : EleTypeE...
EMedType : MedE...
ASDElementReferencesAnalysisErrors
MediaTINGEle
Tool
ToolReferencesDivisionOfLabors
DivisionOfLa...
Tools
DivisionOfLabor
Rule
RuleReferencesDivisionOfLabors
Rules
DivisionOfLaborReferencesHierAsds
HierASDs
DivisionOfLab...
DivisionOfLaborReferencesCommunities
Communities
Community
MediaTEDEle
Subject
SubjectReferencesCommunity
Subjects
Aim
Outcome
OutcomeReferencesNetASDElements
NetASElements
Outcomes
ATModelRootHasMediations
MType : MedType...
TED1Name : String
TED2Name : String
MediationReferencesMediaTINGEles
MediaTINGEles
MediationReferencesMediaTEDEles
MediaTEDEles
MediationReferencesAnalysisErrors
ATModelRootHasDiagColorKeys
DiagColorKeys
DiagColorKey
InfoToDisplay : St...
InfoType : InfoTyp...
ATModelRootHasAnalysisErrors
AEType : AErrorE...
ATModelRootHasContradictions
Contradictions
Contradiction
CIType : ConsidIte...
ContradictionReferencesConsideredASD
ConsideredASD
AsContrads
ContradictionReferencesConsideredAEle
ConsideredAEle
AEsContrads
ContradictionReferencesContradictedASDs
Contradicted...
AsConts
ContradictionReferencesContradictedEles
ElesConts
ContradictionReferencesRelevantASD
RelevantASD
RelAsContrads
ActivityTheoryV5Diagr...
Diagram
ASDShape
CompartmentShape
FillColor : Color
OutlineColor : Col...
TextColor : Color
OutlineDashStyle...
OutlineThickness :...
FillGradientMode...
Decorators
Name
ASDIcon
ExpandCollapseD...
Compartments
Tools
Rules
DoLs
Subjects
Aims
Communities
Outcomes
ASDElementShape
GeometryShape
ToolIcon
RuleIcon
DoLIcon
SubjIcon
AimIcon
CommIcon
OutIcon
Description
ASD2ElementConnector
Connector
MediationShape
UNSETIcon
SAIcon
CAIcon
SCIcon
TINGele
TED1
TED2
relatedASD
ASD2MedConnector
medTINGConnector
medTEDConnector
DoLCreatesASD
OutNetworksASDEle
DoLCommConnector
RulesToDoLsConnector
ToolsToDoLsConnector
SubjCommConnector
DiagColorKeyShape
KeyInfo
ContradictionShape
ContradIcon
ConsideredASD
ConsideredAEle
ContradictedASDs
ContradictedAEles
ContradConsideredItemConnector
ContradictingConnector
ContradRelevantASDConnector