Telelogic

2004

User Group Conference

Innovation Realized

Developing Complex Systems Using DOORS and UML

Michael.Sutherland@galactic-solutions.com

Telelogic

UML and DOORS

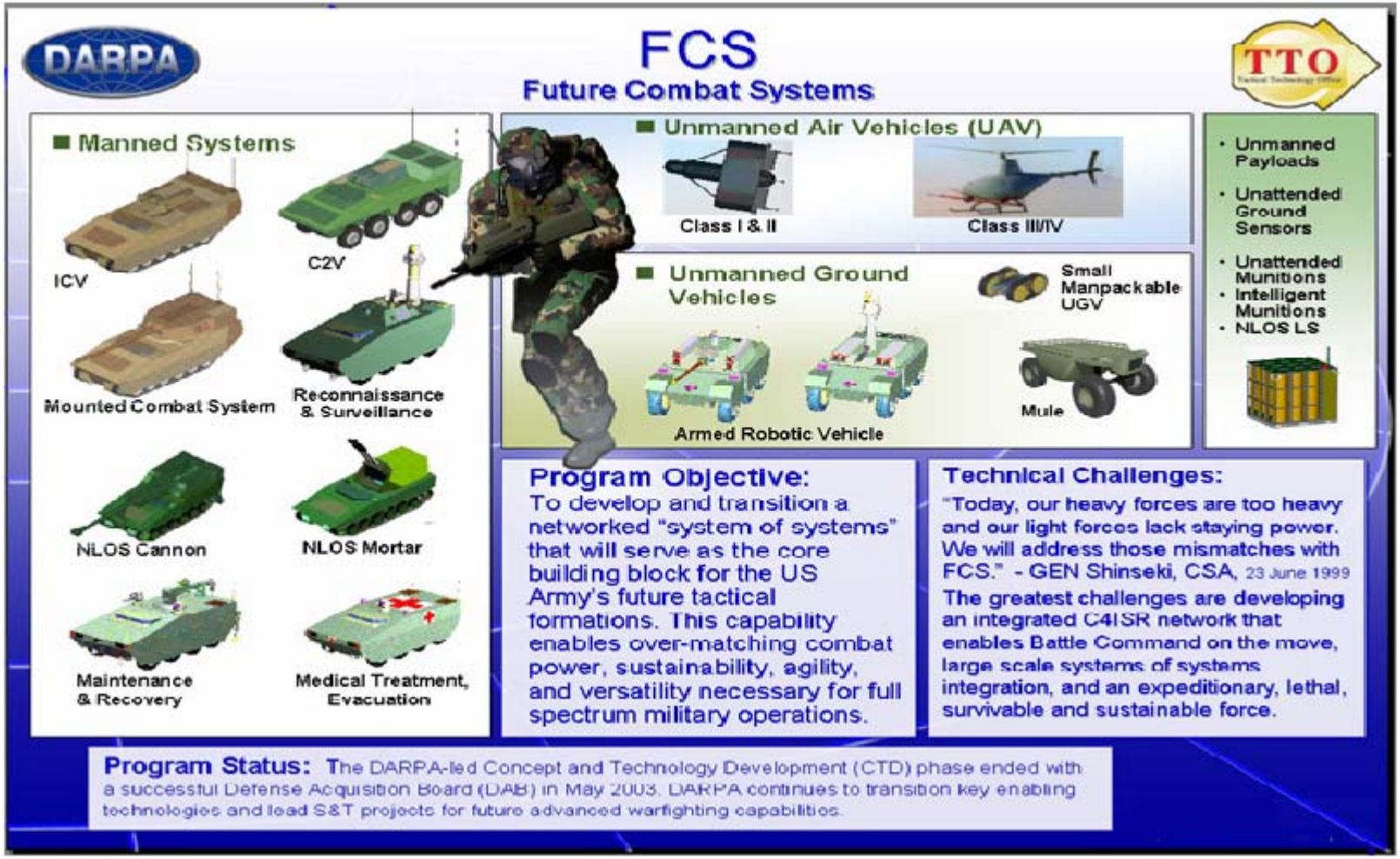
UML™ (www.uml.org)

- The Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a complex system.

DOORS® (www.telelogic.com/doors)

- Telelogic DOORS, the world's leading requirements management tool, is a multi-platform, enterprise-wide system designed to capture, link, trace, analyze and manage changes to information to ensure a project's compliance to specified requirements and standards.

Future Combat Systems (FCS)

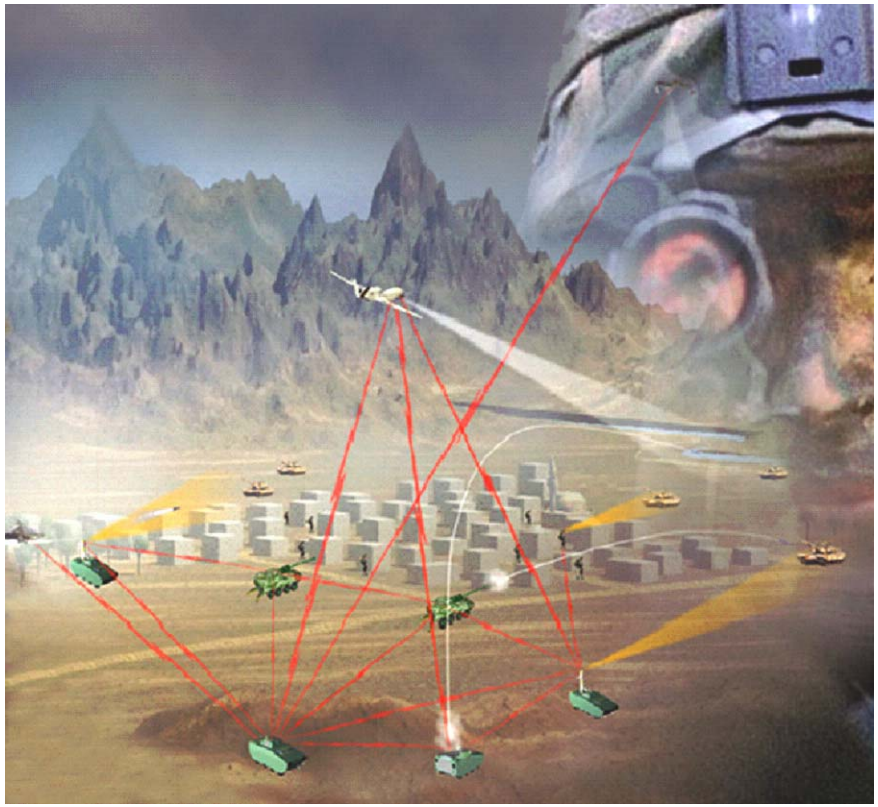


http://www.darpa.mil/tto/PROGRAMS/fcs.html

FCS – Integrated C4ISR Network

FCS is a “System of Systems” (SOS), which includes an integrated C4ISR network.

C4ISR - Command, Control, Communications, Computers, Intelligence, Surveillance, and Reconnaissance

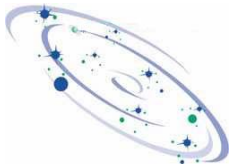


http://www4.army.mil/ocpa/uploads/large/FCSnetwork2004-07-23.jpg

FCS - Participants

The Boeing Company and Science Applications International Corporation (SAIC), are the FCS Lead System Integrators (LSI).

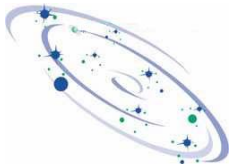
General Dynamics Land Systems (GDLS) and United Defense Limited Partnership (UDLP) were selected by the FCS LSI to form an integrated design team for the Manned Ground Vehicle (MGV) portion of the FCS program.



Modeling Approach and Tool Set

- The FCS LSI chose a “one model” approach to development, where the entire FCS “System of Systems” (SOS) will be modeled in UML, and all suppliers will provide UML models which will integrate with the SOS.

- The FCS LSI selected Telelogic DOORS as the tool for requirements management, and selected IBM Rational Rose as the tool for UML modeling.



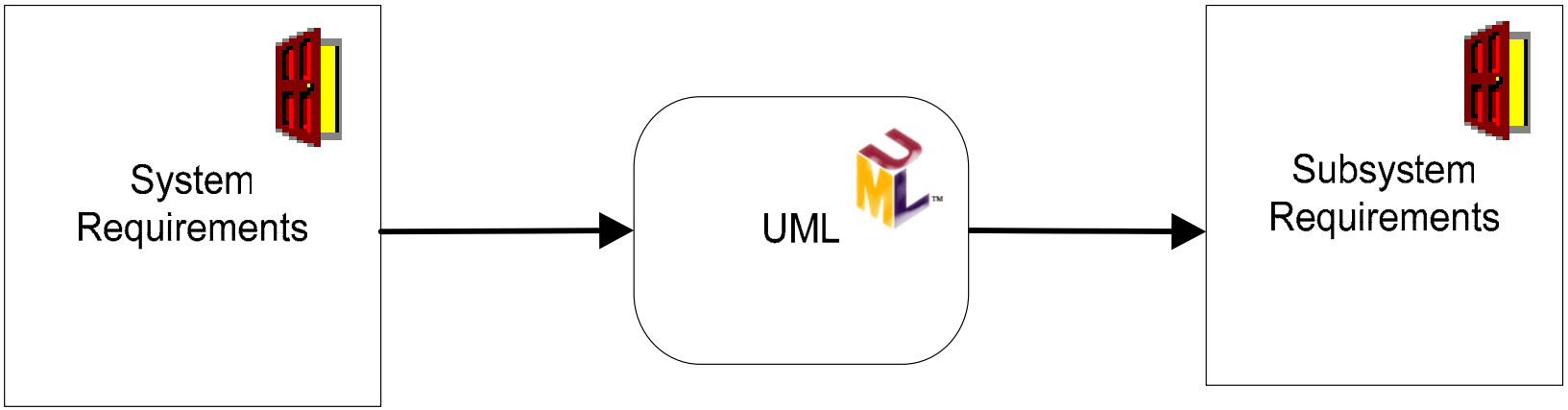
Need for Development Process

- UML does not prescribe a specific process for modeling, and DOORS does not prescribe a specific process for requirements management.

- Given the need for specific processes, GDLS and UDLP have developed a process for MGV Systems Requirements Analysis which includes processes to document system requirements in DOORS, and to model system behavior using UML.

- Among the process requirements is the need to establish traceability from customer provided System Requirements in DOORS to derived UML modeling elements. Traceability is also established for all requirements derived from UML model elements.

UML-DOORS Systems Requirements Analysis Process



The steps in the process steps are numbered 1-18.

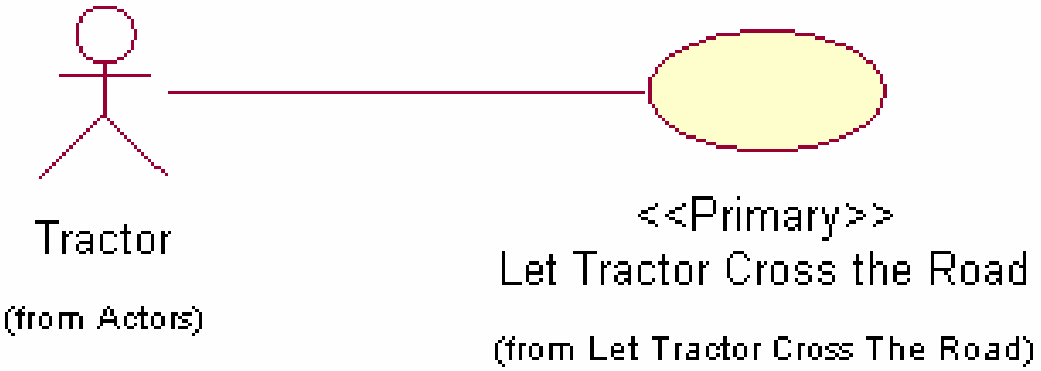
Terminology is explained in bulleted form.

System Requirements

1. Develop and evaluate System Level Requirements in DOORS.

Use Case Diagrams

- Use Case Diagrams describe the functionality of the system from perspective of those that are outside it and interacting with it.



Use Case – Elements and Process

Elements:

- Actor – An entity external to the system that plays a role by interacting with the system.

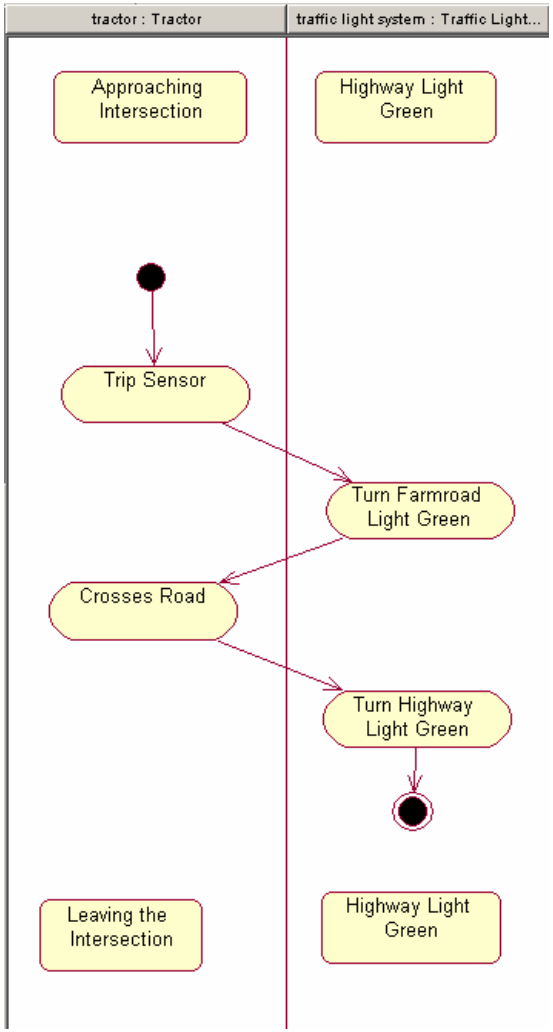
- Use Case – Functional behavior that the system will perform to provide value to the Actor.

Process:

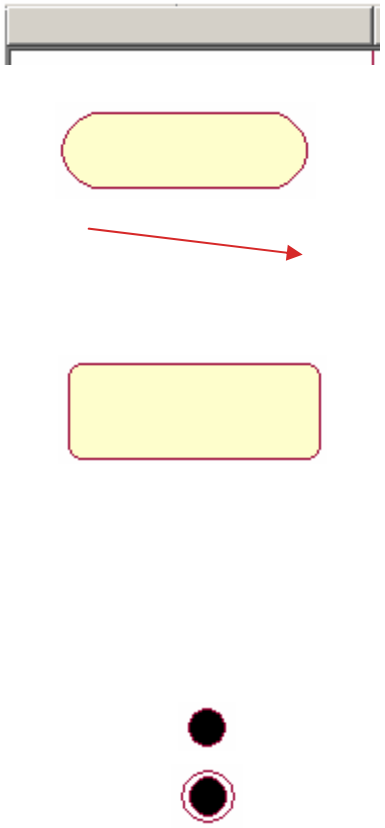
2. Determine Actors and develop Use Cases from System Requirements.
3. Document the relationships between Uses Cases and Actors on Use Case Diagrams.
4. Develop Traceability from Use Cases to the System Requirements they were derived from.

Activity Diagrams

- Activity Diagrams are used to decompose Use Cases by describing the flow of events that comprise a Use Case.
- Similar to flowcharts.

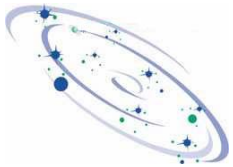


Activity Diagram - Elements



Elements:

- Swimlanes – Partitions that show which Objects are responsible for performing each Activity.
- Activity – A tangible unit of work.
- Transition – Flow of control between Activities. A transition occurs when an Activity is completed.
- State - A state is a named condition or situation in the life of an object, that lasts for some finite amount of time, during which the object satisfies some condition, performs some activities and/or actions, or waits for some event.
  - Precondition – Initial State of Object
  - Postcondition – Final State of Object



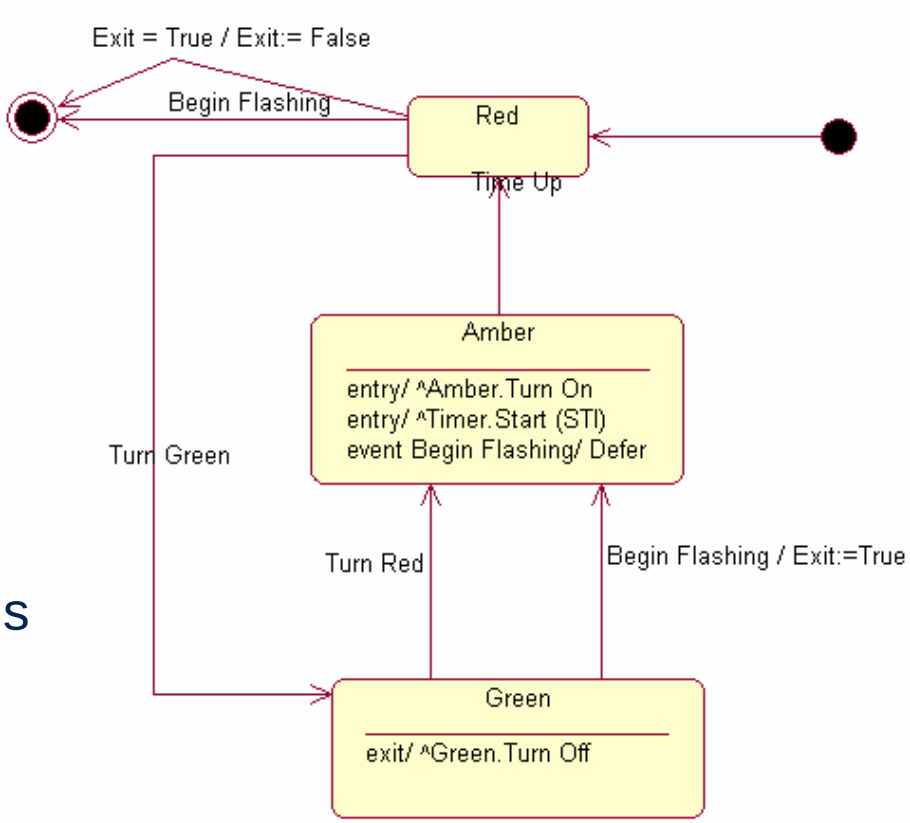
Activity Diagrams - Process

Process:

5. Determine the Activities that need to be performed to accomplish each Use Case.
6. Determine the Objects that will perform each Activity.
7. Assign each Object to a Swimlane on an Activity Diagram. A Swimlane shall represent exactly one Object.
8. Assign each Activity to a Swimlane on an Activity Diagram. A Swimlane may contain multiple Activities.
9. Document and the flow of control between the Objects as Transitions between Activities on Activity Diagrams.
10. Document the preconditions and postconditions for each Object as States in each Swimlanes on Activity Diagrams.

Statechart Diagrams

- A Statechart Diagram is used to show events that cause a class to transition from one state to another state, and the actions that result from a class changing state.

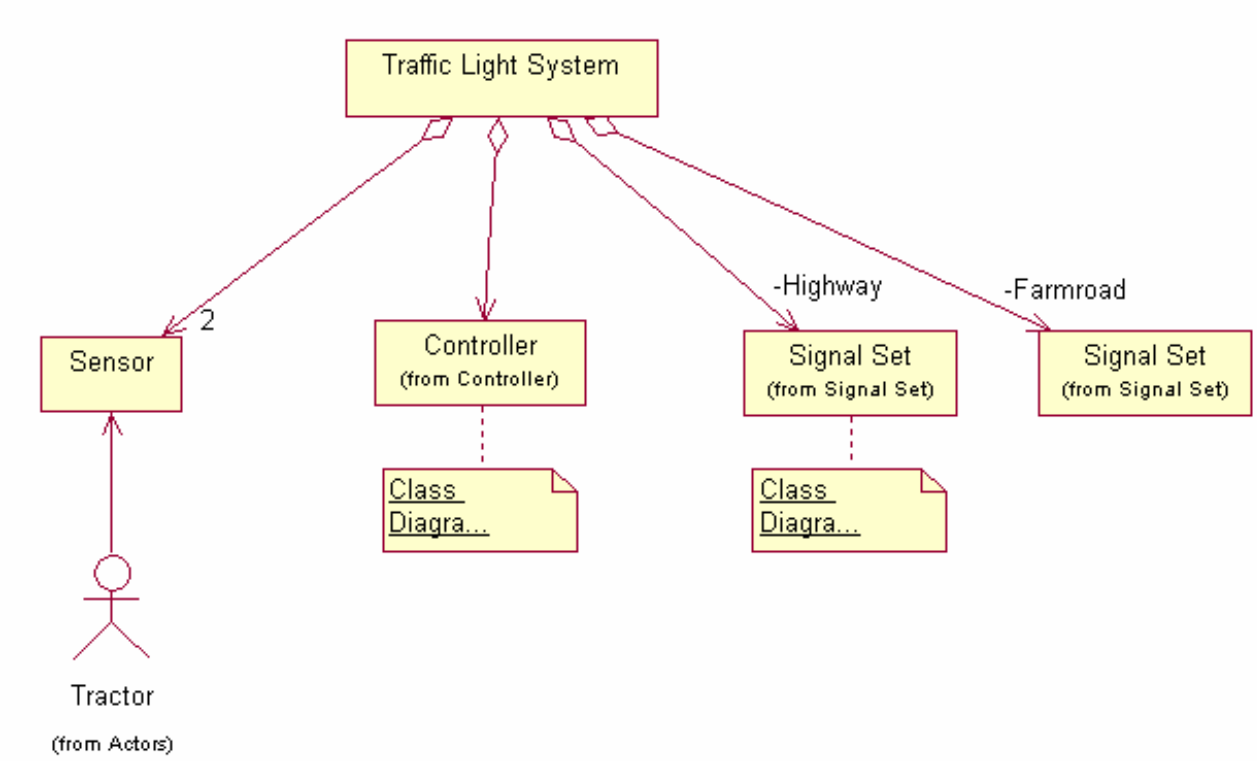


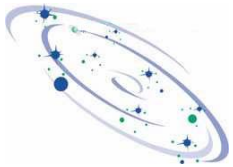
Process:

11. Develop Statechart Diagrams (Sequence of states, stimuli, and response).

Class Diagrams

- A Class Diagram documents relationships (Association, Aggregation, Composition, Dependency, Inheritance, Realization) between Classes.





Class Diagrams - Process

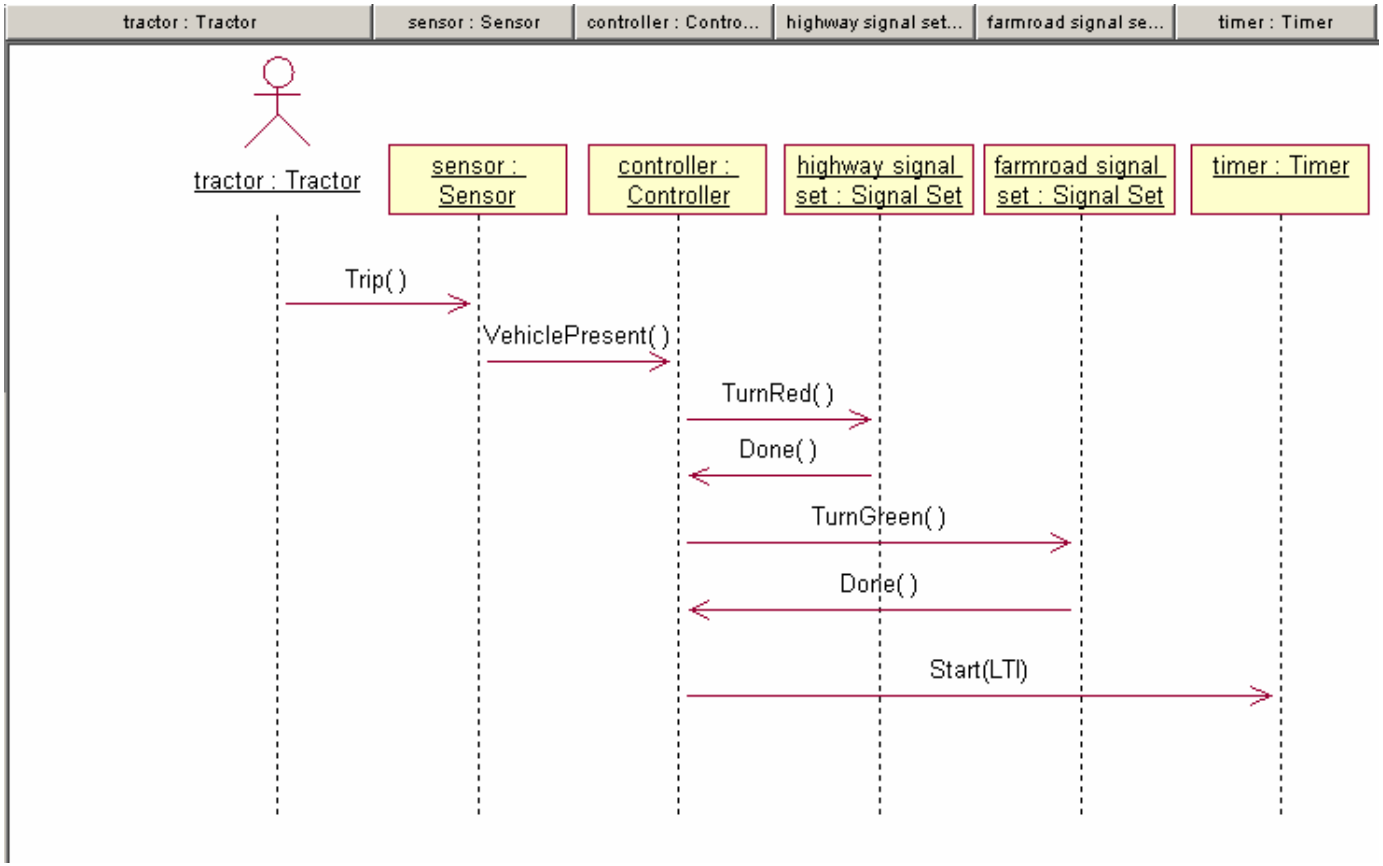
- There is no single step called “Develop Class”, because Classes may be developed at any step. The following are all Classes that were developed in previous steps:
  - An Actor on a Use Case Diagram
  - An Object on an Activity Diagram (associated with a Swimlane)
  - An Object on a Sequence Diagram (associated with a Lifeline)
- Each Class has Attributes (or Properties), that define the data structure of the Class.
- Each Class has Operations (or Methods), that define the functions of the Class.

Process:

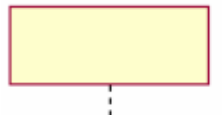
12. Document relationships between Classes on Class Diagrams.

Sequence Diagrams

- Sequence Diagrams describe the behavior of Objects, including the sequence of Messages between Objects.



Sequence Diagram – Elements and Process



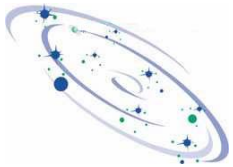
Operation(Parameters)

Elements:

- Object – A Class represented by vertical dashed “Lifeline”
- Message – Communication (interaction) between Objects. Each messages is bound to an Operation.
- Operation – Function of a Class

Process:

13. Add Objects to Sequence Diagrams (breakdown activities).
14. Add Messages between Objects in Sequence Diagrams.
15. Document the sequence over time of the messages between Objects on Sequence Diagrams.
16. Bind each Message to an Operation, document on Sequence Diagrams.



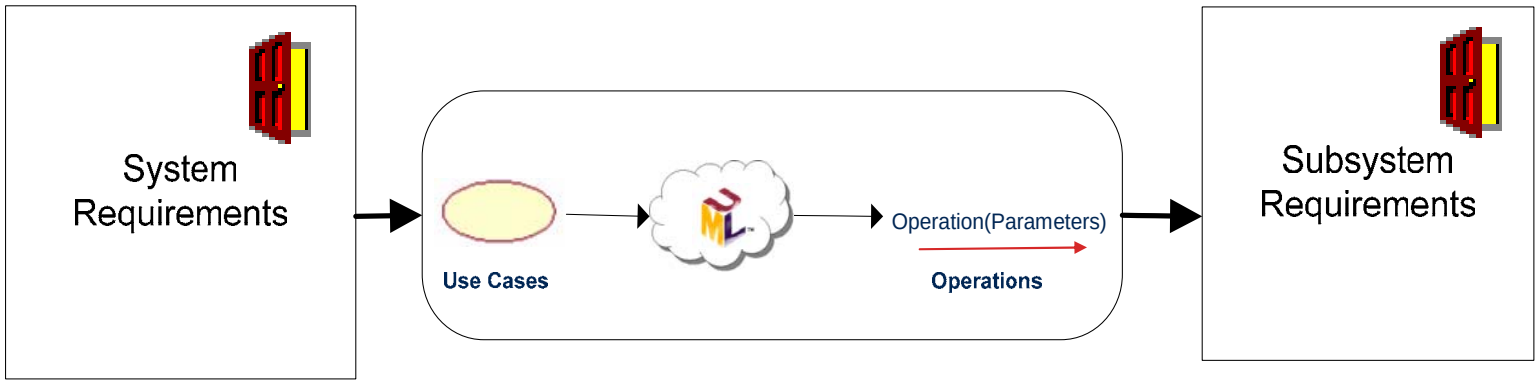
Operational Contracts

- An Operational Contract is defined for each Operation, to give precise detail about the preconditions that are necessary for the Operation to be executed, and the postconditions that result after the Operation is executed.
- Postconditions may include the creation and deletion of Class instances, modification of Class Attributes, and the formation or breaking of associations between Classes.
- An Operation may use parameters (data), and may return data.

Process:

17. Develop Operational Contracts
18. Derive Subsystem Requirements from Operational Contracts

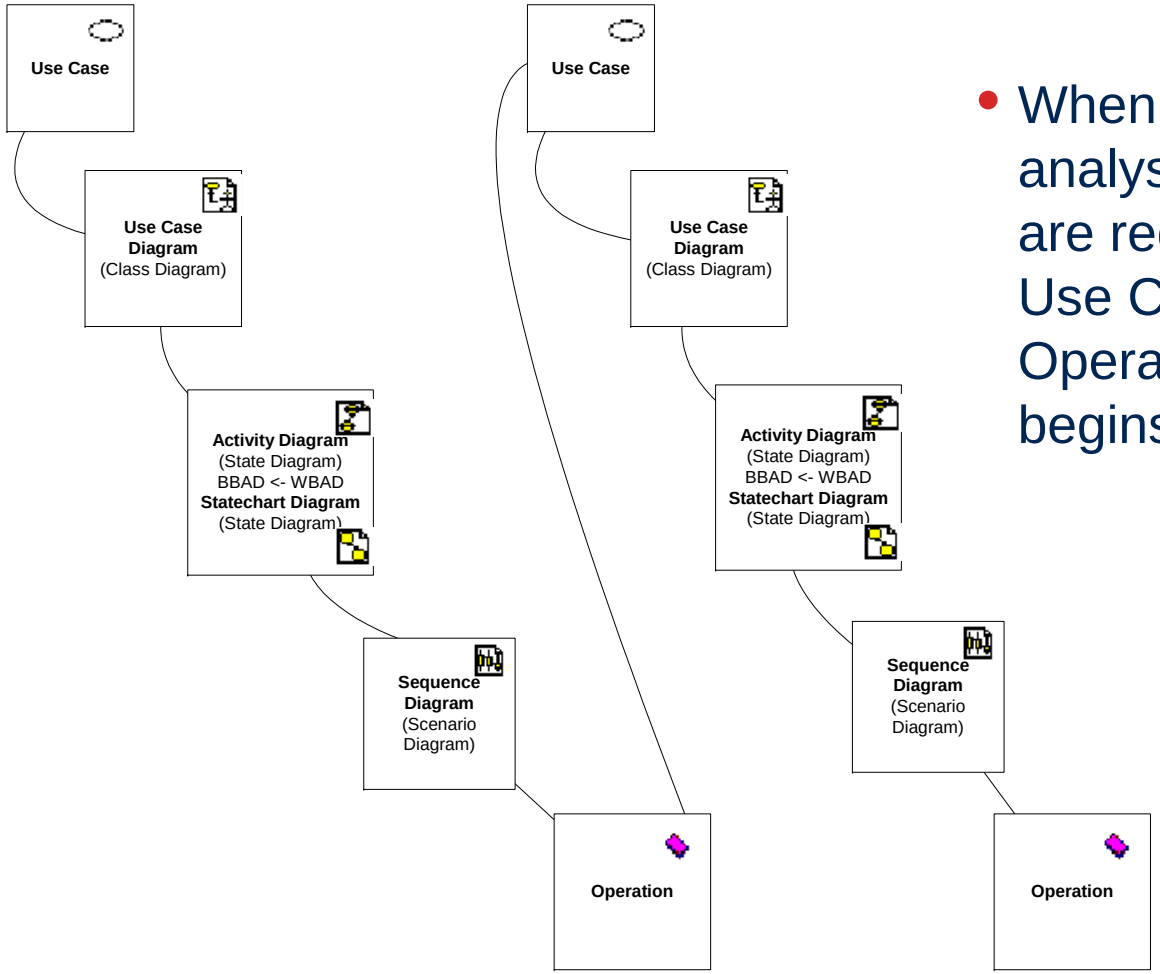
Generating Requirements from Operational Contracts



- New Subsystem Requirements are derived from Operational Contracts, documented in DOORS, and traced back to the Operations in the UML surrogate Module.

- An Operation may generate multiple Subsystem requirements, and a single Subsystem requirement may be generated from multiple Operational Contracts.

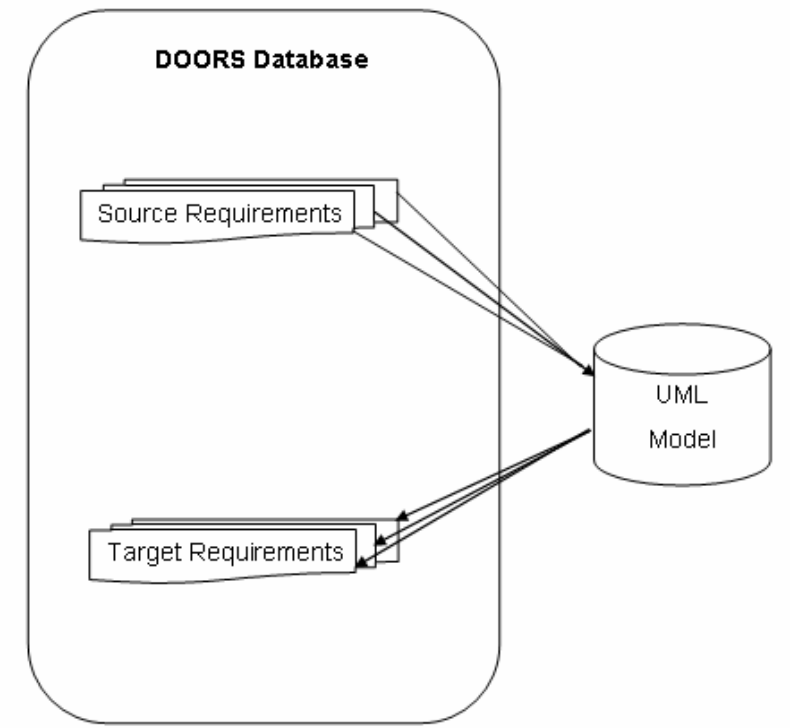
Analysis and Decomposition

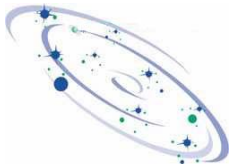


- When further levels of analysis and decomposition are required, more detailed Use Cases are derived from Operations, and the process begins again.

Traceability between UML and DOORS

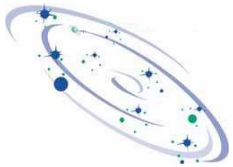
- Traceability must be established:
  - Between Use Cases and the System Requirements they were derived from
  - Between Subsystem Requirements and the Operations they were derived from





RoseLink - Evaluation

- Telelogic has developed a product called “Rose Link” to accomplish requirements traceability between the two tool sets.
- GDLS evaluated Telelogic RoseLink v2.8, and found that even though it is a very capable integration, it did not fit well into the FCS MGV System Requirements Analysis process for the following reasons:
  - No graphical representations of UML diagrams were made available in DOORS.
  - Some UML elements and diagram types were not represented in DOORS in any form.
  - Knowledge of the complex UML Browser hierarchy structure was not represented in the DOORS Module Explorer view.
  - Knowledge of the relationships between UML elements was not well represented.



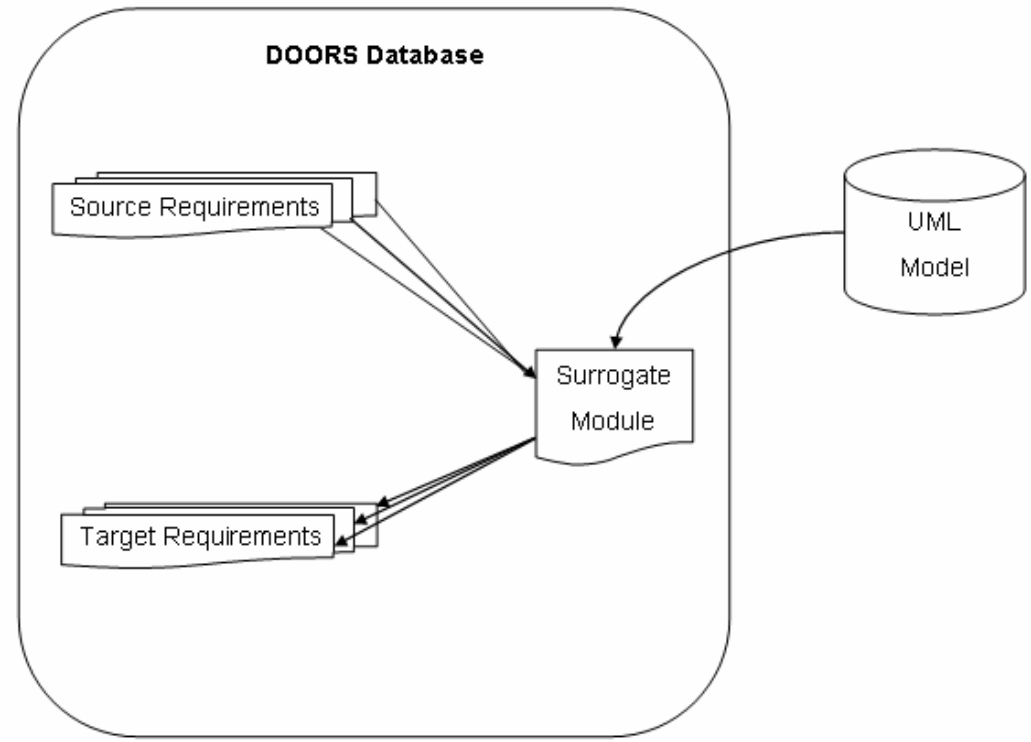
RM-Int - Features

- Because of the critical importance of establishing traceability throughout the System Requirements Analysis process, GDLS has developed a custom integration between DOORS and Rose called “RM-Int” (Requirements/Modeling Integration).

- The Requirements/Modeling Integration that was developed as part of the UML DOORS System Requirements Analysis Process has the following features:

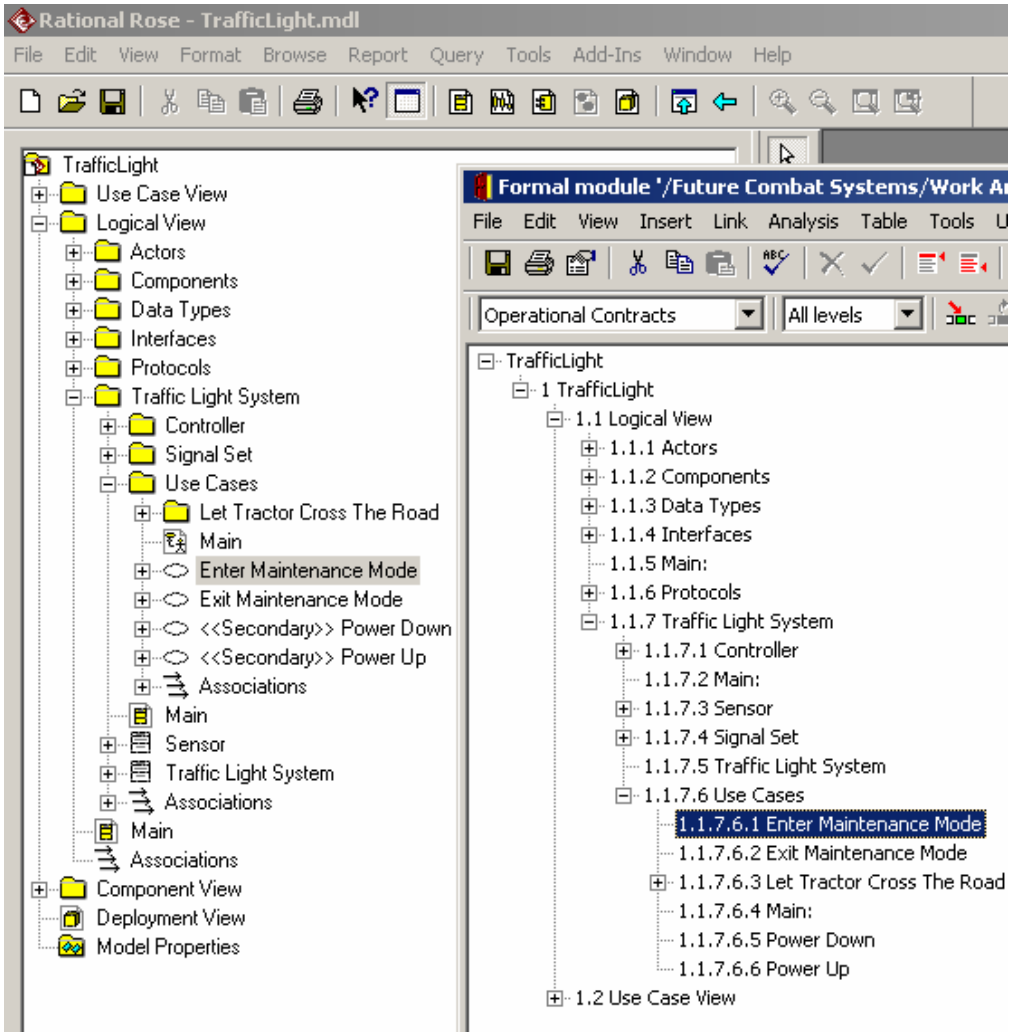
RM-Int – Surrogate Module

- Makes the model more widely available for consumption.
- Daily synchronizations allow for timely communication of changes.
- Filtering and sorting allow for custom views.



RM-Int - UML Hierarchy Browser

- A complex UML model in is organized into UML elements called Packages (folders).
- Packages allow related elements to be grouped in UML browser hierarchy.
- The integration duplicates the UML browser hierarchy in the DOORS surrogate module.



RM-Int – UML Element Icons

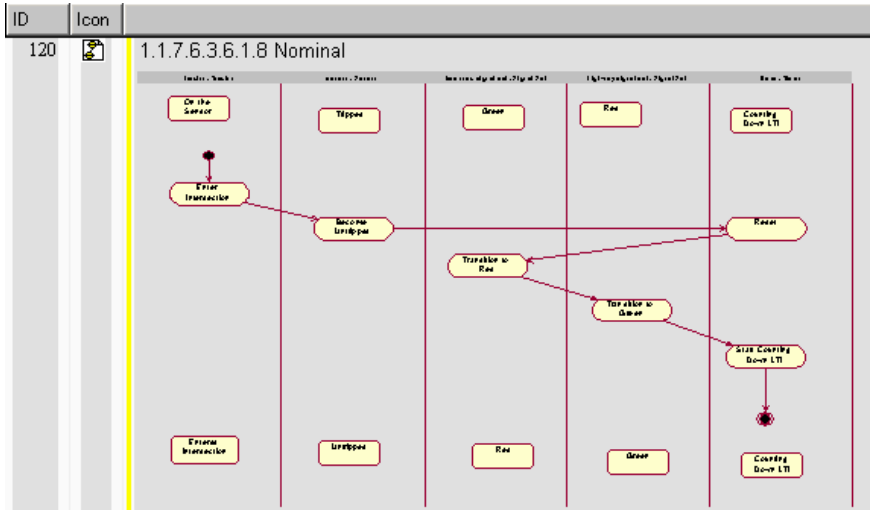
- Every UML element may have a stereotype and a corresponding Icon.
- Displaying this Icon in a DOORS text attribute allows for a concise display and quick identification of UML elements.
- Requirement Engineers may not be familiar with these Icons, so the stereotype name is also available in a DOORS attribute.

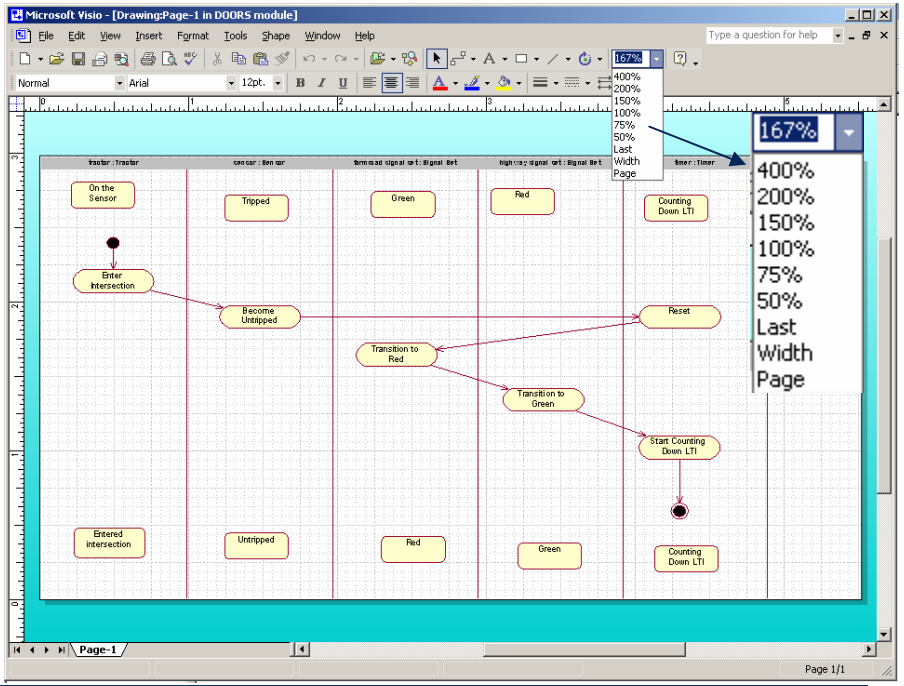
Icon	Element Stereotype
	Activity
	ActivityDiagram
	Actor
	Attribute
	boundary
	Category
	Class
	ClassDiagram
	CollaborationDiagram
	control
	ControlledCategory
	entity
	FinalState
	InitialState

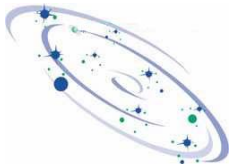
Icon	Element Stereotype
	Interface
	Model
	Operation
	PackageDiagram
	PrivateAccess
	SequenceDiagram
	State
	StateMachine
	StatechartDiagram
	UnloadedCategory
	UseCase
	UseCaseDiagram

RM-Int – UML Diagrams

- Diagrams serve to communicate much of the UML content. Without adequate representation in the surrogate UML module, it would not be possible to comprehend the model.
- The UML integration brings in a Visio *bitmap* representation of each diagram. Visio was chosen because of the availability of pan and zoom functions.

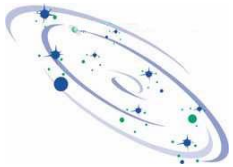






RM-Int – Deleted Model Elements

- As the UML Model changes, model elements will be deleted.
- The DOORS application will not allow corresponding objects to be deleted until the incoming traceability links are removed.
- Such elements in the surrogate module are marked as deleted by the integration.
- This mechanism allows requirements engineers who have generated requirements from the model to be informed that model elements they were previously utilizing has been deleted by a modeler, while preserving traceability.
- Elements deleted from the UML model that do not have traceability relationships established are deleted and purged from the surrogate module.

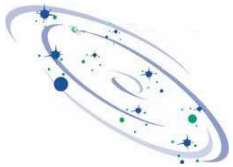


RM-Int - Metrics

- Metrics of the DOORS Surrogate Module are possible without additional scripting. For example, the following table lists counts of UML element types for a given day:

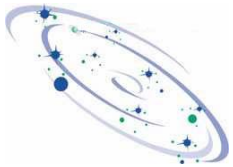
Activity	3272
Attribute	106
Category	2193
Class	967
Operation	937
State	2704
StateMachine	549
UseCase	539
UseCaseDiagram	224
ClassDiagram	658
PackageDiagram	316
SequenceDiagram	272
ActivityDiagram	563
StatechartDiagram	43

- Traceability metrics are also possible. For example, a count of Operations that derive other Requirements could be taken, given the fact that in such cases the derived Requirements will be linked to the Operations.



RM-Int – Model Relationships (1/2)

- The Packaging standards used for complex models may dictate that related elements reside in different parts of the UML hierarchy, including residing in different packages.
- To increase navigability, DOORS links are created in the UML surrogate from UML diagrams to the UML elements that are on the diagrams.
  - Use Case Diagram -> Use Case
  - Use Case Diagram -> Actor
  - Class Diagram -> Class
  - Activity Diagram -> Activity
  - Activity Diagram -> State
  - State Diagram -> State
  - Sequence Diagram -> Operation

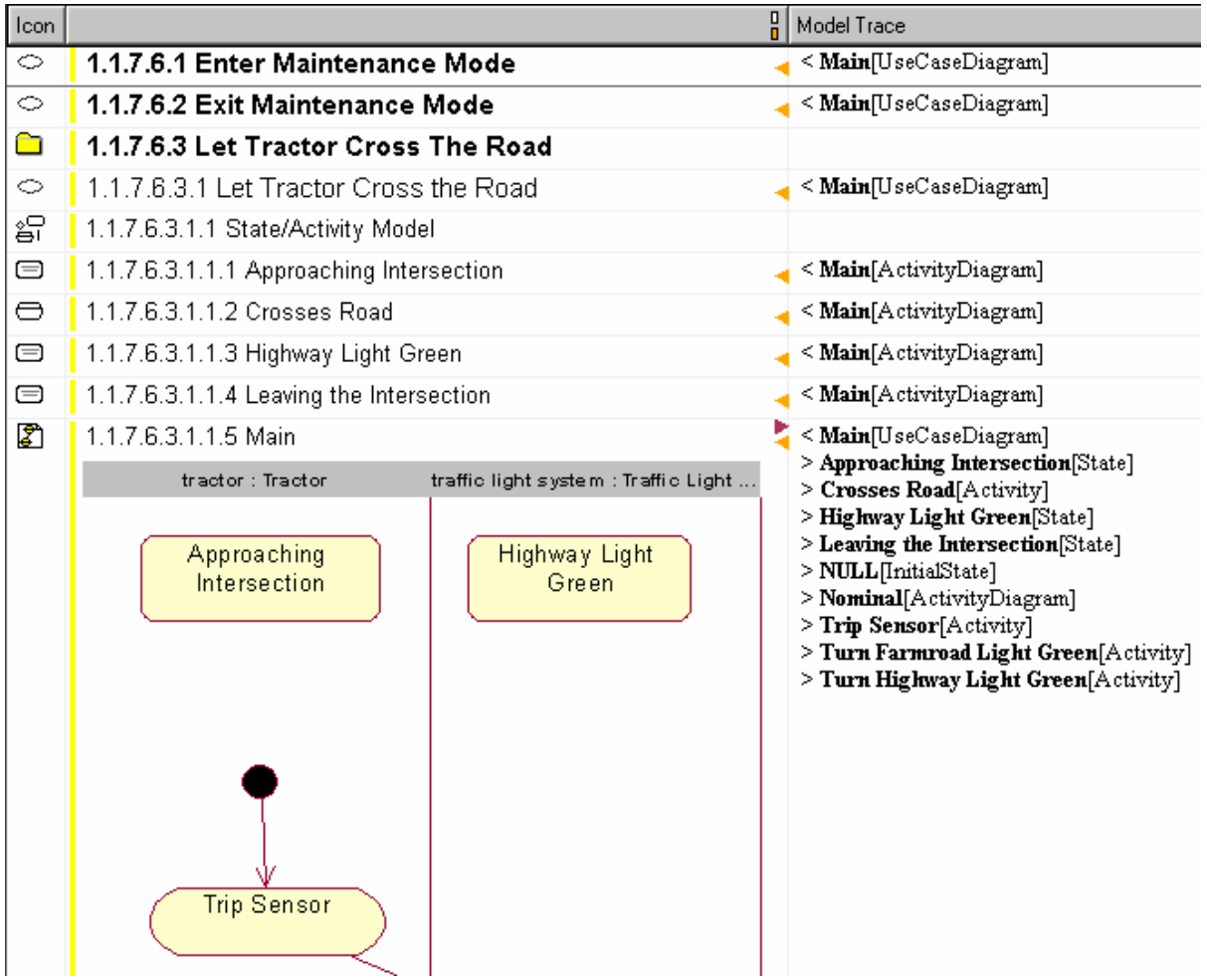


RM-Int – Model Relationships (2/2)

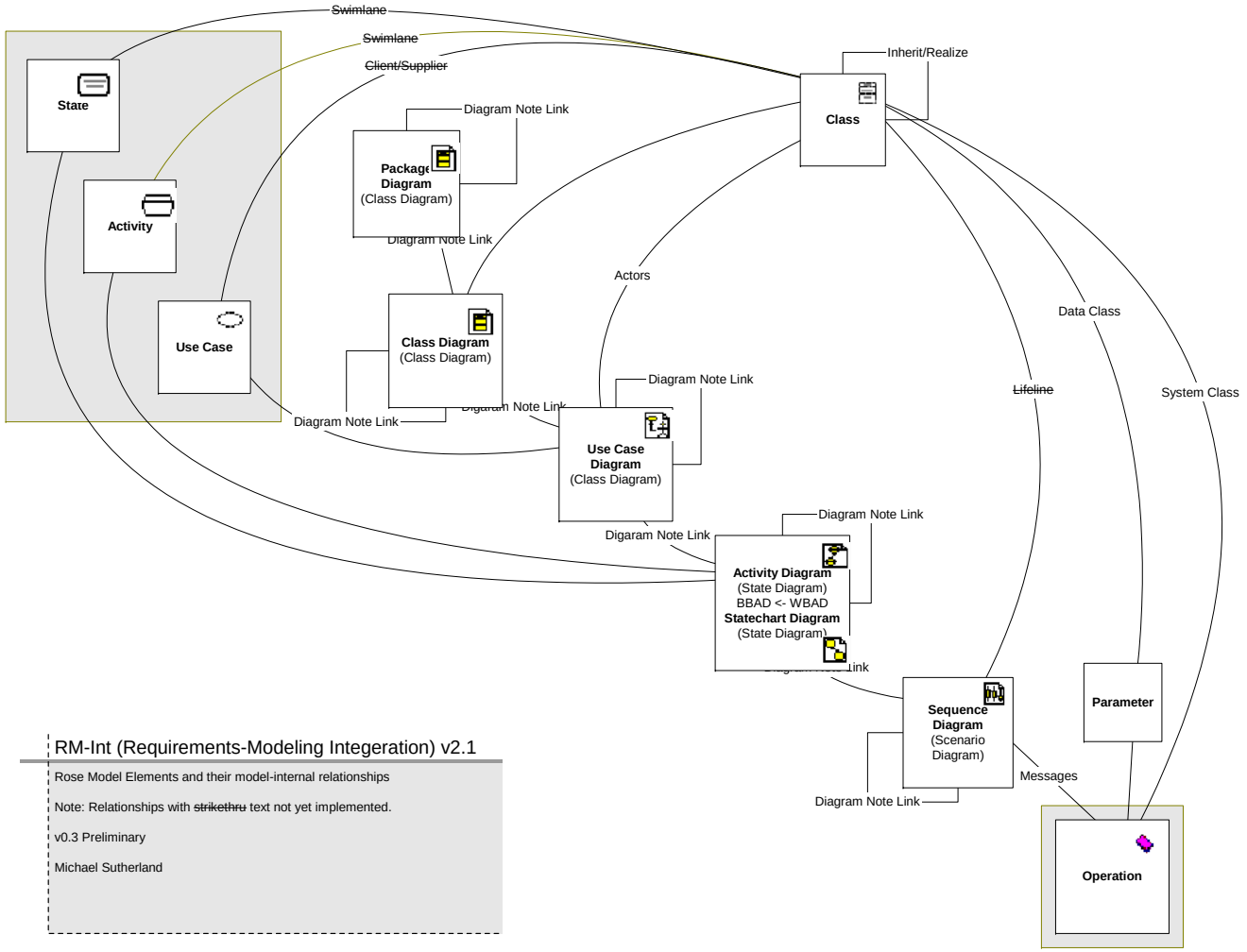
- UML also supports linking via an element called a Note, and specifically allows one UML diagram to link to any other diagram. For example, an Activity diagram will be note-linked to the Use Case diagram that it was generated from.
  - Diagram-> Diagram
- Non-diagram elements also have relationships that need to be navigated:
  - Operation -> Class
  - Operation(Parameter[1-n]) -> Class[1-n]
  - Class(InheritRelation[1-n]) -> Class[1-n]
  - Class(RealizeRelation[1-n]) -> Class[1-n]

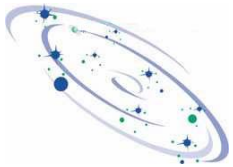
RM-Int – Model Relationship Trace

- The “Model Trace” Column on the far right shows detail about the model-internal relationships for each UML element.



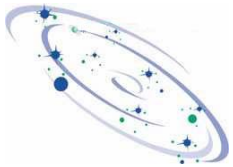
RM-Int – Model Internal Relationships





Benefits of DOORS-Rose Integration

- The main benefits this effort has achieved are:
  - Requirements Engineers now access the updated UML model in a consistent manner, where before they received arbitrary exports from the UML modeling tool at the modeler's discretion.
  - Requirements Engineers have the ability to establish traceability from UML to source requirements and derived requirements in a manner consistent with other traceability procedures for the project.
  - Modelers can now better comprehend the how a change to the UML model will impact the system being developed.



Conclusion

- Modeling a complex system with UML has proven to be a daunting task.
- Traditional requirements engineers, while having a wealth of domain knowledge, have had difficulty expressing this knowledge in UML.
- Given this, the project decided to add experienced UML modelers to facilitate the task.
- Experienced UML modelers, while having a deep knowledge of expressing concepts in UML, often enter the project with no domain knowledge.
- Bridging the gap between those with domain knowledge, and those with UML knowledge, has been one of the main challenges faced when developing a System Requirements Analysis Process.