Developing Software architectures using UML and formal methods

Artificial Intelligence makes the programmer more intelligent not the computer.

Robert van Halder
October 5, 2004

Contents

- Analyses and design
- Architecture
- Cleanroom Software Engineering
- Formal validation
- Recommendations
- Questions

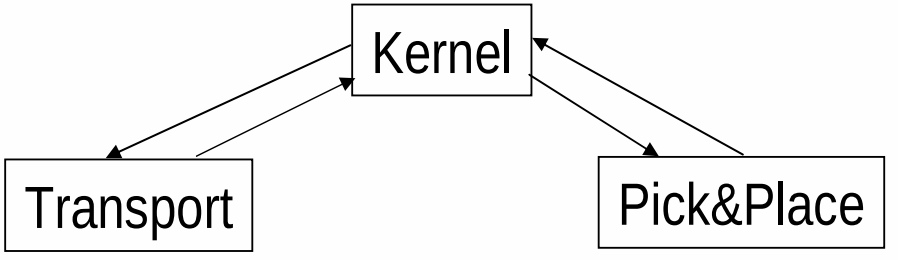
Analysis and design [overview]

- Specify the requirements and behavior of the system.
  - Define mechanisms for threading, communication etc.
- Analyze the behavior of the system.
  - Make use cases, activity diagrams, sequence diagrams or state diagrams.
  - Sequence diagrams can be used to describe mechanisms especially for communication between modules.
- Define all interfaces

Note: Try to prevent to go into too much detail in an early stage!!

Analysis and design [Example 1]

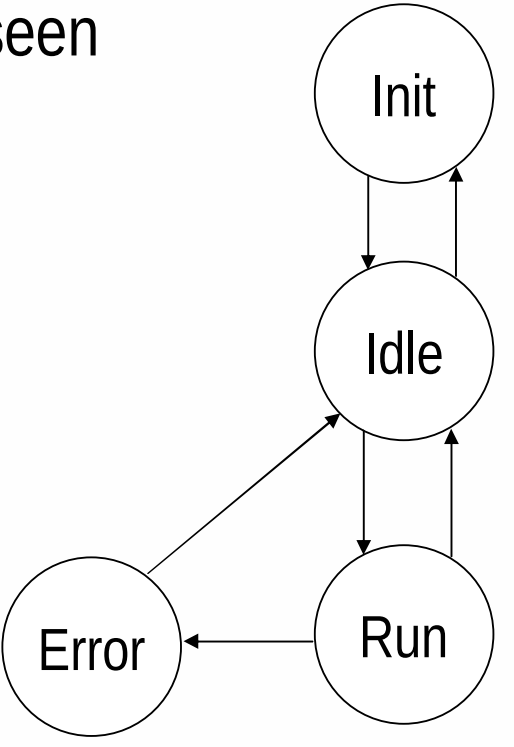
- Module: Transport system for PCB's
  - As part of a whole system.



- Requirements:
  - System state behavior
  - Transport behavior
  - Testability for the modules
  - SMEMA protocol for board transfer from and to other systems in the production line

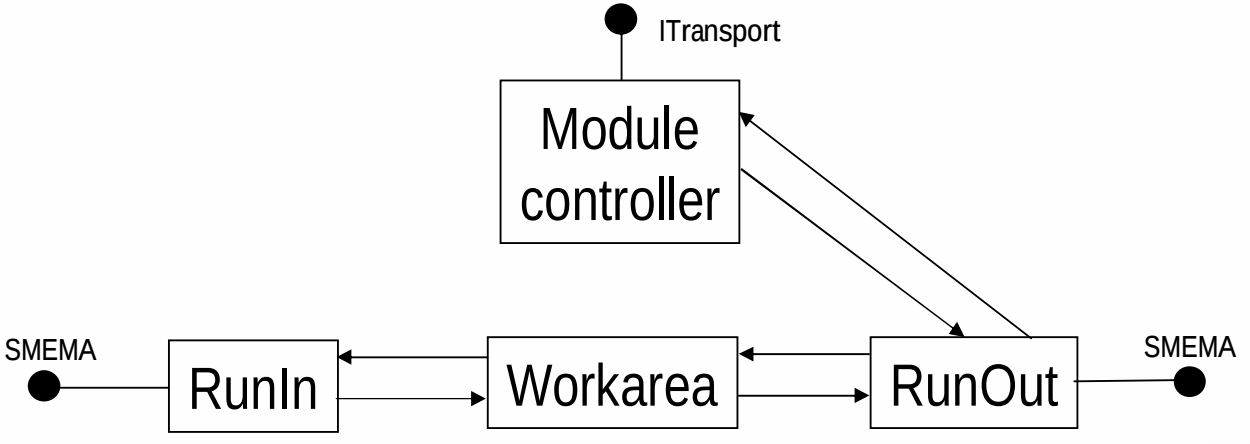
Analysis and design [Example 2a]

- Identify the interface (ITransport) as seen by the user of the whole system
  - IN: Initialize, Terminate, Start, Stop, Recover.
  - Out: Initialized, Terminated, Started, Stopped, Error, Recovered.



Analysis and design [Example 2]

- Choose an architectural pattern for transport
  - The starting point is the pipes and filters pattern:
    - Every module adds something to a board just like in the example of Buschmann.
    - Every module only knows its Left and Right neighbor
- Choose the main functions:
  - ModuleController, RunIn, Workarea, RunOut



Analyses and design [considerations]

- Every pattern has its liabilities. Try to find a way to overcome these liabilities.
  - The pipes and filters pattern biggest liability is Error handling.
- Note that it's best to chose a pattern that is fitting a real world problem.
  - Something the designers are familiar with, or that they can imagine very good.
  - Introduce a new name for the pattern if this helps the designers to imagine the use of the pattern. (It sometimes takes some time before all designers see the similarities with existing patterns.)

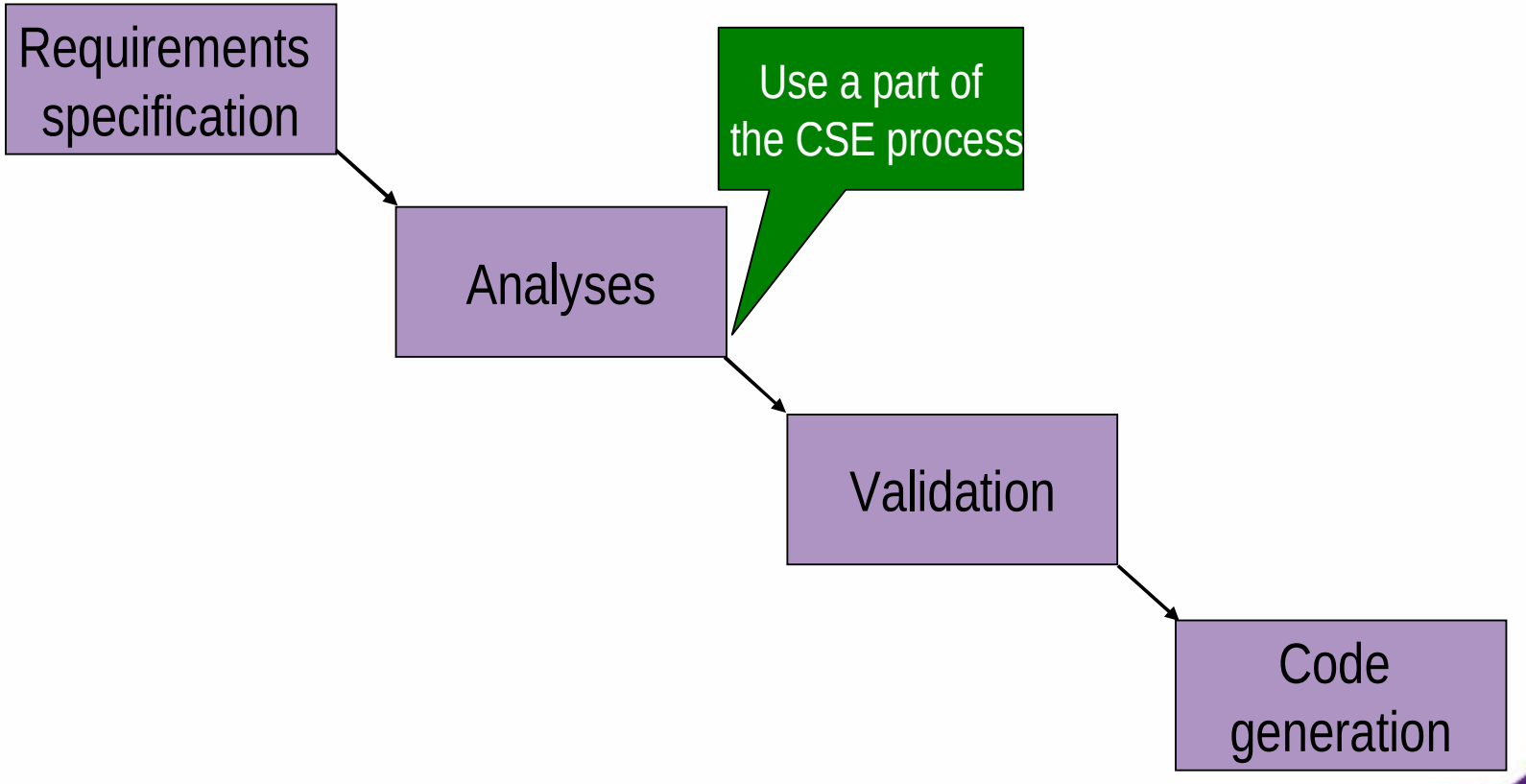
Architecture [General characteristics]

- In a lot of OO projects UML is used to specify system design and behavior.
- UML has been developed keeping office applications in mind.
- UML 2.0 has only a bit more support for behavior specific applications. (Timing diagrams are added)
- The state diagrams are supported in UML but the use of it is very limited.
- Instead of the using state diagrams in UML use the analyses part of Cleanroom Software Engineering (CSE).

Architecture [2]

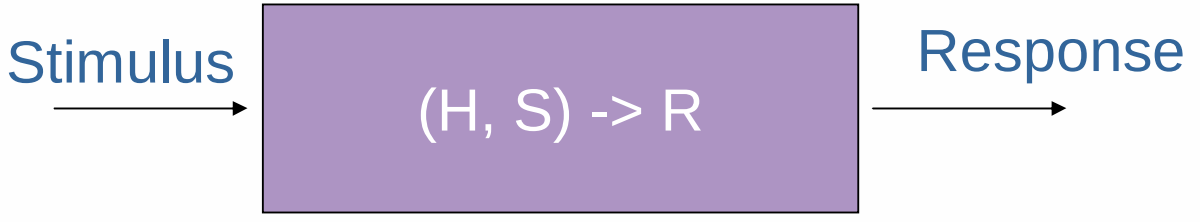
- To perform a proof of concept for an architecture the main focus are non-functional requirements.
  - The functional requirements are mostly covered directly
- Behavioral requirements are not always proven.
  - UML is not very useful for specifying behavior.
  - Cleanroom Software Engineering can be used to specify and proof that the system design behaves correctly.
  - Tools can be used to validate that the system behavior is correct before the implementation starts.

Cleanroom Software Engineering



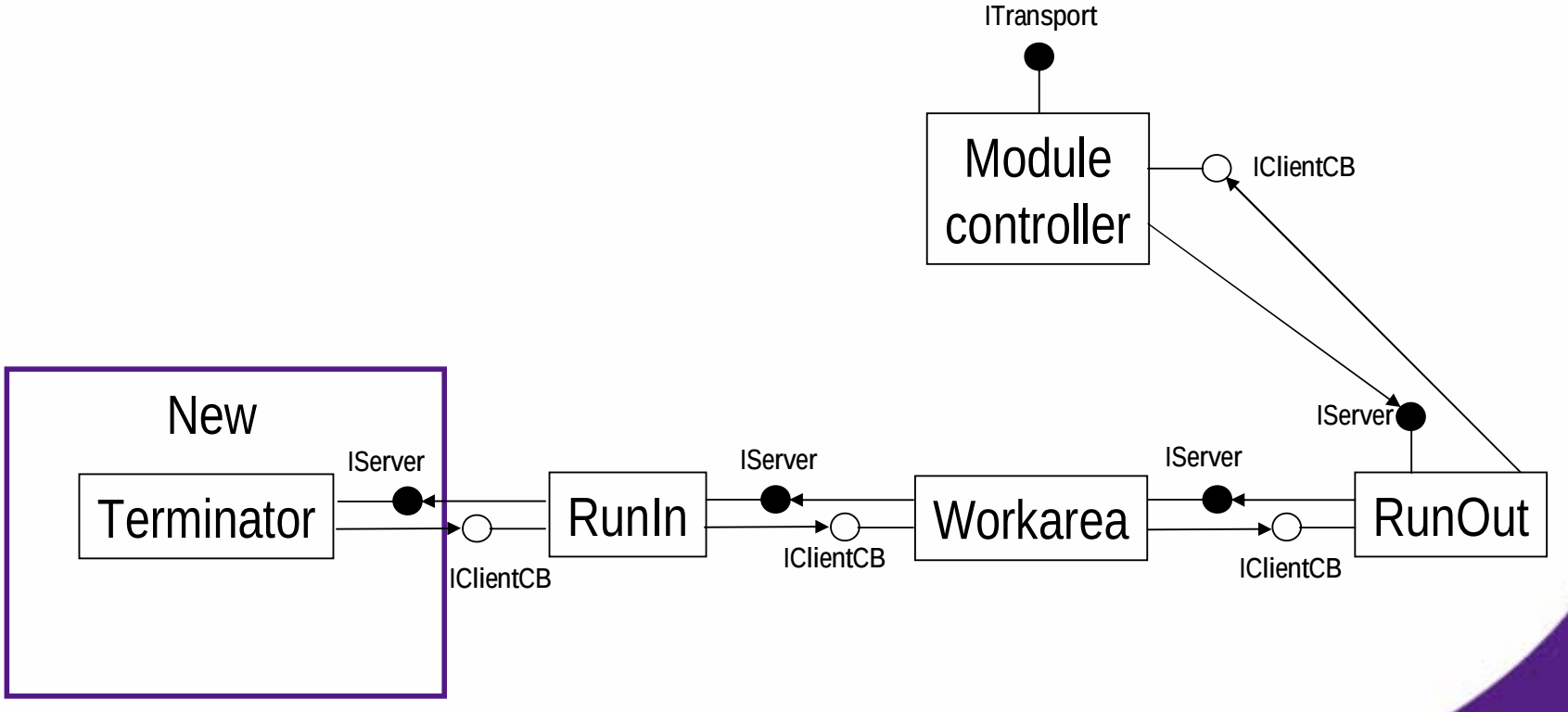
Cleanroom fundamentals [1]

- CSE uses the sequence-based specification method.
- Start with the specification of a black box.
- Continue with the specifications of a state box.
  - This is the inside out view of a software part (white box)
- Finish with the specifications of a clear box.
  - This is the formal specification of a function or method.



Interfaces [Example 1]

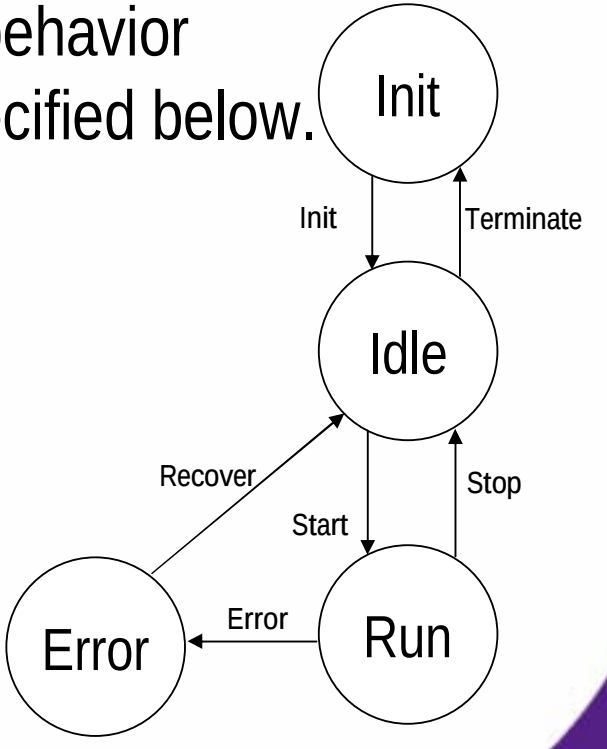
- Determine internal interfaces

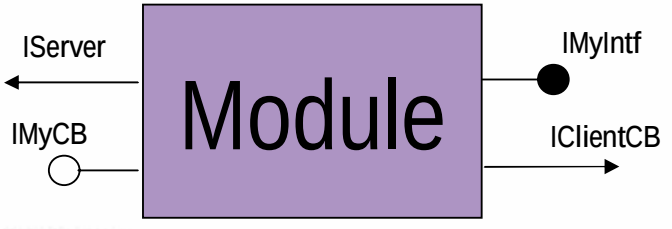


Interfaces [Example 2]

- Prepare for a formal analysis of the behavior of the module using the interface specified below.

IMyInt:	IMyCB:
• Init	• Initialized
• Start	• Started
• Stop	• Stopped
• Recover	• Error
• Terminate	• Recovered
	•Terminated





Black box specification [1]

Title	Generic Pipeline Element Spec (version 35)								External State	
No	**Stimulus**	**Condition**	**Response**		**Equ**	**Explanat**	**Trace**		**From**	**To**
0:<>						done	**UNINT**			
1:<Cl.rqInitialize(NoOp[bChangeOverRequested=false; bOOC=false; bFinishing=false; bCOR=false])>						done	**Initializing**			
53:<Cl.rqInitialize,A_Gpe.TransitionComplete(Cl.ntInitCompleted_Ok[bCOR=false])>						done	**Idle**			
54	Cl.rqInitialize		Illegal		-		D0	53	Idle	
55	Cl.Terminate		NoOp		0		D3	53	Idle	UNINT
56	Cl.SetProducingMode		NoOp		53		D5	53	Idle	Idle
57	Cl.SetModuleEngineeringMode		NoOp		57		D4	53	Idle	Idle_Eng
58	Cl.SetCalibrationMode		NoOp		53		D5	53	Idle	Idle
59	Cl.SetSimulationMode		NoOp		53		D5	53	Idle	Idle
60	Cl.RetrieveProcessProgram		NoOp		53		D27	53	Idle	Idle
61	Cl.rqValidateProcessProgram		Cl.ntValidateProcessProgram		53	No errors are	D7	53	Idle	Idle
62	Cl.rqValidateSetUp		Cl.ntValidateSetup		53		D7	53	Idle	Idle
63	Cl.ValidateChangeOver		NoOp		53		D29	53	Idle	Idle
64	Cl.rqPrepareStartProduction		NoOp		64		D8; D6	53	Idle	Preparing
65	Cl.rqStartProduction		Illegal		-		D0	53	Idle	
66	Cl.rqPauseProduction		Illegal		-		D0	53	Idle	
67	Cl.rqFinishProduction		NoOp	bFinishing=true	53	Fire and	D22	53	Idle	Idle
68	Cl.rqAbort		Illegal		-		D0	53	Idle	
69	Cl.FinalizeAbort		NoOp	bCOR=false	53	"Remove	D14	53	Idle	Idle
70	Cl.rqAllowErrorRecovery		Illegal		-		D0	53	Idle	
71	Cl.rqRecover		Illegal		-		D0	53	Idle	
72	Cl.rqInitiateChangeOver		NoOp		53	Fire and	D13	53	Idle	Idle
73	Cl.DoRollingChangeOver		Illegal		-		D0	53	Idle	
74	Cl.TestEmpty		NoOp		53	Check	D32	53	Idle	Idle
75	Cl.FinishCalibration		NoOp		53	Store the	D32	53	Idle	Idle
76	Cl.rqForcedStop		Illegal		-		D0	53	Idle	
77	A_Gpe.Error		Illegal		-		D0	53	Idle	
78	A_Gpe.ProductionFinished		Illegal		-		D0	53	Idle	
79	A_Gpe.ChangeOverRequest		Illegal		-		D0	53	Idle	
57:<Cl.rqInitialize,A_Gpe.TransitionComplete,Cl.SetModuleEngineeringMode(NoOp[])>						done	**Idle_Eng**			
64:<Cl.rqInitialize,A_Gpe.TransitionComplete,Cl.rqPrepareStartProduction(NoOp[])>						done	**Preparing**			

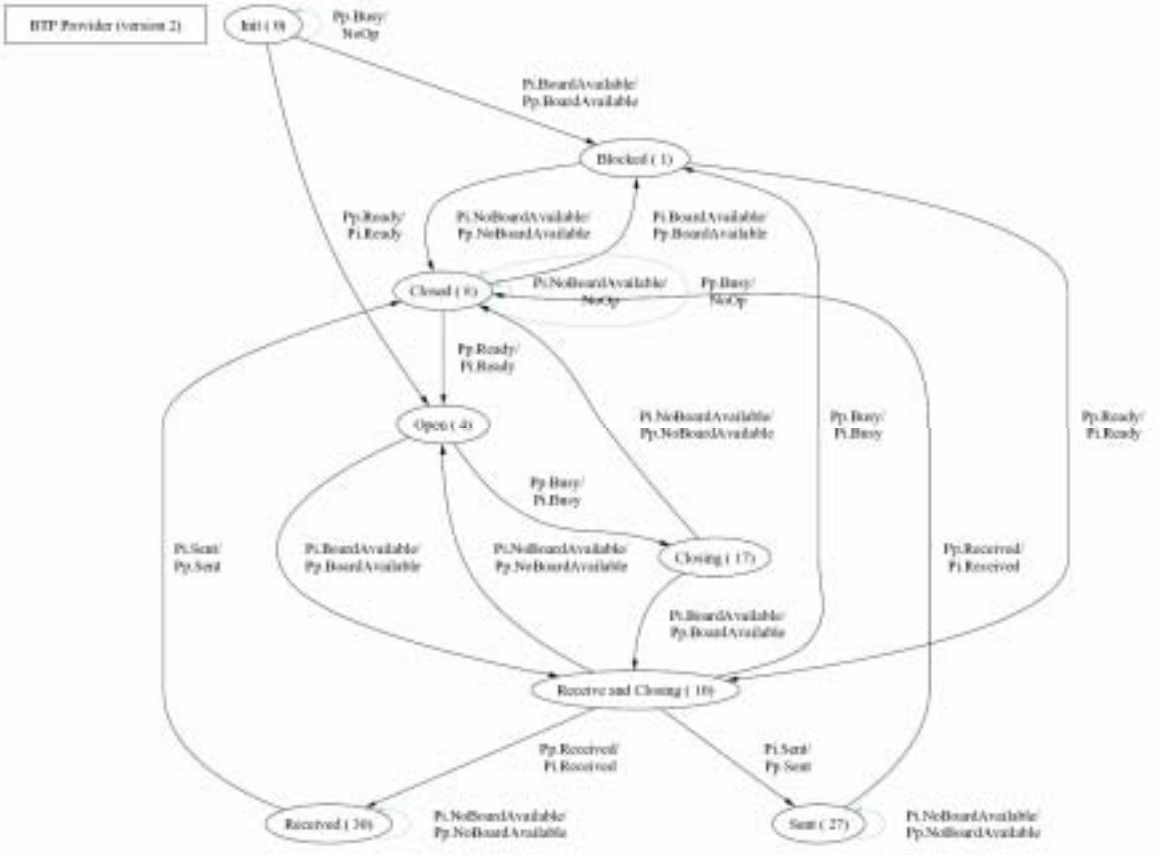
There is a total of 17 different states.

White box specification [2]

Title	Generic Pipeline Element (version 35)							External State		Server State		Modu
No	Stimulus	Condition	Response		Equ	Trace		From	To	From	To	From
0:<> 1:<Cl.rqInitialize(Srv.rqInitialize[bPFin=false; bCOR=false; bOOC=false; eVal=none])> 78:<Cl.rqInitialize,Srv.ntInitializationOK(Mod.rqInitialize[])> 145:<Cl.rqInitialize,Srv.ntInitializationOK,Mod.ntInitializationOK(Cl.ntInitCompleted_Ok[])>						UNINT(UnInit, UnInit) Initializing(Initializing, UnInit) Initializing(Idle, Initializing) Idle(Idle, Idle)						
163	Cl.rqInitialize		Illegal		-	D0	145	Idle		Idle		Idle
164	Cl.Terminate	&eVal=none	Mod.Terminate; Srv.Terminate		0	D3	145	Idle	UNINT	Idle	UnInit	Idle
165	Cl.Terminate	&eVal!=none	Illegal		-	D0	145	Idle		Idle		Idle
166	Cl.SetProducingMode	&eVal=none	Mod.SetProducingMode; Srv.SetProducingMode		145	D5	145	Idle	Idle	Idle	Idle	Idle
167	Cl.SetProducingMode	&eVal!=none	Illegal		-	D0	145	Idle		Idle		Idle
168	Cl.SetModuleEngineeringMode	&eVal=none	Mod.SetModuleEngineeringMode; Srv.SetModuleEngineeringMode		168	D4	145	Idle	Idle_Eng	Idle	Idle_Eng	Idle
169	Cl.SetModuleEngineeringMode	&eVal!=none	Illegal		-	D0	145	Idle		Idle		Idle
170	Cl.SetCalibrationMode	&eVal=none	Mod.SetCalibrationMode; Srv.SetCalibrationMode		145	D5	145	Idle	Idle	Idle	Idle	Idle
171	Cl.SetCalibrationMode	&eVal!=none	Illegal		-	D0	145	Idle		Idle		Idle
172	Cl.SetSimulationMode	&eVal=none	Mod.SetSimulationMode; Srv.SetSimulationMode		145	D5	145	Idle	Idle	Idle	Idle	Idle
173	Cl.SetSimulationMode	&eVal!=none	Illegal		-	D0	145	Idle		Idle		Idle
174	Cl.RetrieveProcessProgram	&eVal=none	Srv.RunProcessProgram; Mod.RunProcessProgram		145	D27	145	Idle	Idle	Idle	Idle	Idle
175	Cl.RetrieveProcessProgram	&eVal!=none	Illegal		-	D0	145	Idle		Idle		Idle
176	Cl.rqValidateProcessProgram	&eVal=none	Mod.rqValidateProcessProgram	eVal=PP_M	145	D7	145	Idle	Idle	Idle	Idle	Idle
177	Cl.rqValidateProcessProgram	&eVal!=none	Illegal		-	D0	145	Idle		Idle		Idle
178	Cl.rqValidateSetUp	&eVal=none	Mod.rqValidateSetup	eVal=SU_M	145	D7	145	Idle	Idle	Idle	Idle	Idle
179	Cl.rqValidateSetUp	&eVal!=none	Illegal		-	D0	145	Idle		Idle		Idle
180	Cl.ValidateChangeOver	&eVal=none	Mod.ValidateChangeOver; Srv.ValidateChangeOver		145	D29	145	Idle	Idle	Idle	Idle	Idle
181	Cl.ValidateChangeOver	&eVal!=none	Illegal		-	D0	145	Idle		Idle		Idle
182	Cl.rqPrepareStartProduction	&eVal=none	Mod.rqPrepareStartProductionPRE		182	D8; D6	145	Idle	Preparing	Idle	Idle	Idle
183	Cl.rqPrepareStartProduction	&eVal!=none	Illegal		-	D0	145	Idle		Idle		Idle
184	Cl.rqStartProduction		Illegal		-	D0	145	Idle		Idle		Idle
185	Cl.rqPauseProduction		Illegal		-	D0	145	Idle		Idle		Idle
186	Cl.rqFinishProduction	&eVal=none	Mod.rqFinishProduction; Srv.rqFinishProduction		145	D22	145	Idle	Idle	Idle	Idle	Idle
187	Cl.rqFinishProduction	&eVal!=none	Illegal		-	D0	145	Idle		Idle		Idle
188	Cl.rqAbort		Illegal		-	D0	145	Idle		Idle		Idle
189	Cl.FinalizeAbort	&eVal=none	Mod.FinalizeAbort; Srv.FinalizeAbort		145	D14	145	Idle	Idle	Idle	Idle	Idle

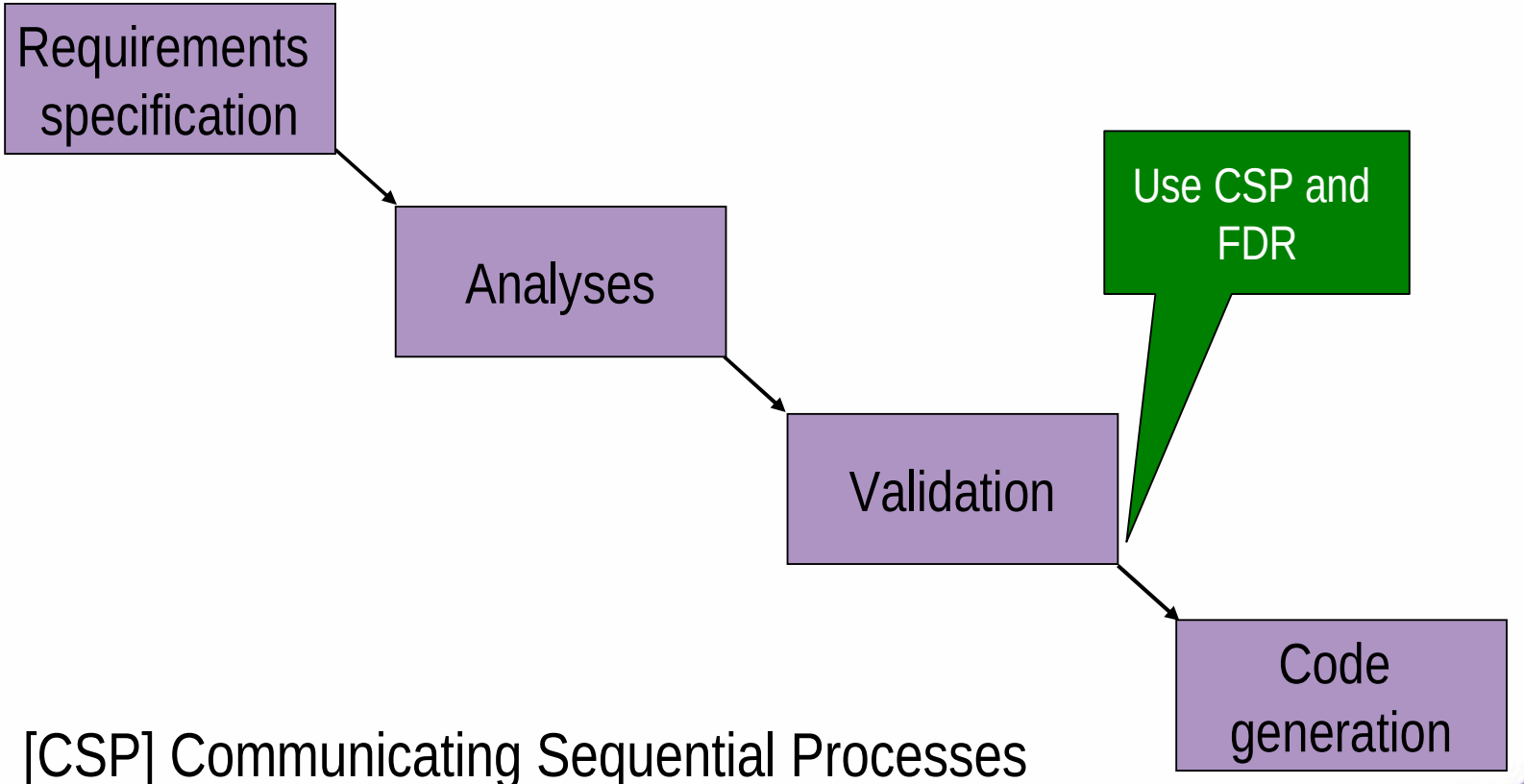
There is a total of 54 different states.

Black/white box specification [3]



This is from another specification to show the graphical representation of the cleanroom sheets.

Formal validation



[CSP] Communicating Sequential Processes
[FDR] Failures-Divergence Refinement

Formal validation [tools]

- Mathematical way of describing parallel and sequential processes (by Hoare)
- FDR2 is a refinement checker for establishing properties of models expressed in CSP.
- FDR is a tool used to validate the results of the Cleanroom analyses that where formally specified with CSP.
- Source : http://www.usingcsp.com/cspbook.pdf
  http://www.fsel.com

Formal validation [Experience]

- FDR is already used within Assembléon to validate the synchronization protocol within the AX by Formal Systems.
- Also for the Generic Pipeline within the AX-Kernel and AX-Transport software Imtech.

Recommendations [1]

- UML is very useful to specify OO designs
- Cleanroom is very useful for starting projects
- The combination of sequence diagrams and Cleanroom sheets is very powerful
- Use Cleanroom always in combination with CSP and FDR
- Cleanroom is most useful in combination with code and test generation

Recommendations [2]

- Using Cleanroom, CSP and FDR requires a lot of mathematical skills from the developers. It is not realistic to expect this from the developers.
  - Also, the skills reduce very fast when they are not used!!!
- To get the best of the whole Formal Software Development is best to hire specialists to join the team.

Questions