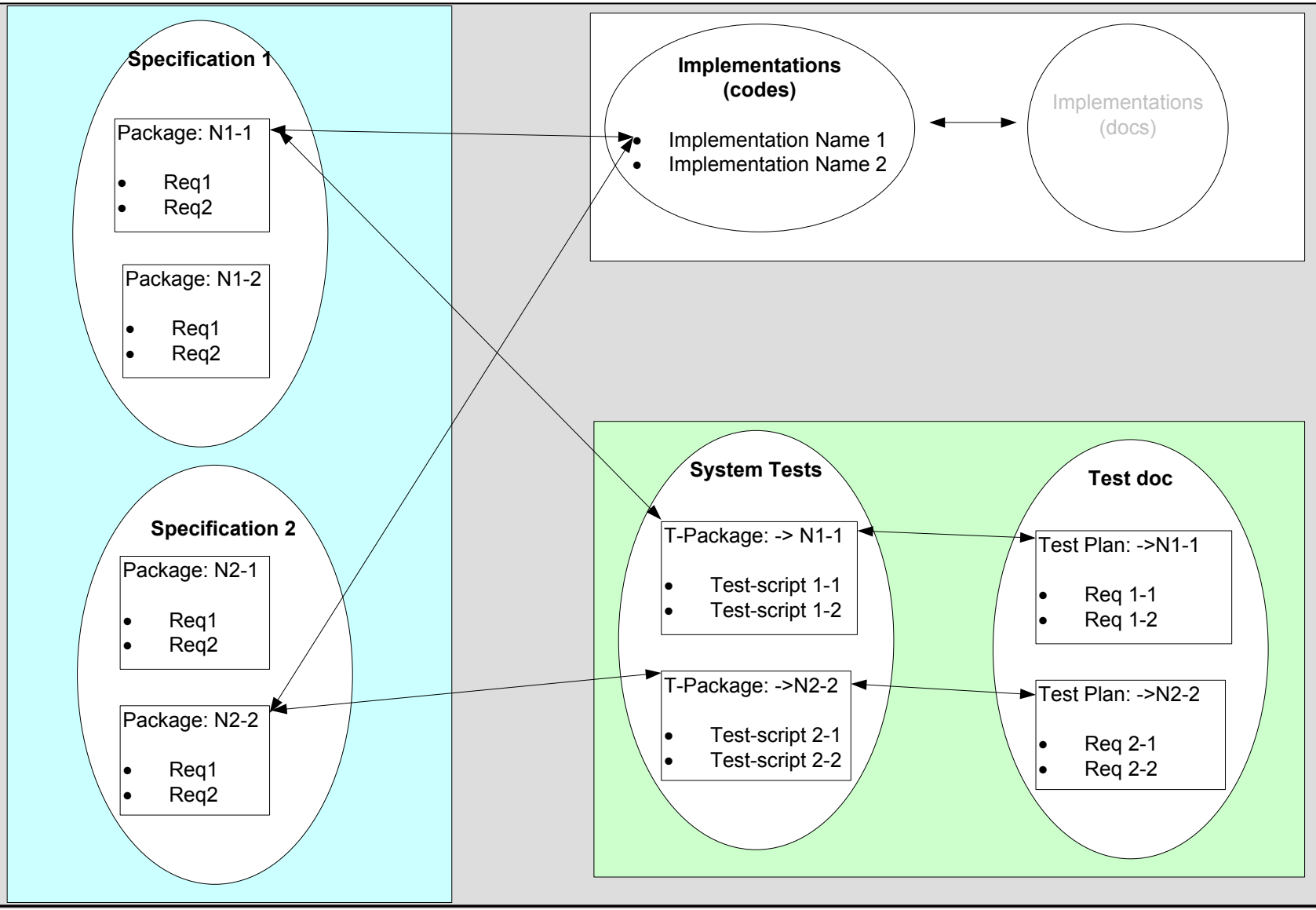
Formal Modeling in Test Development at G&D

G&D, 17.11.2005

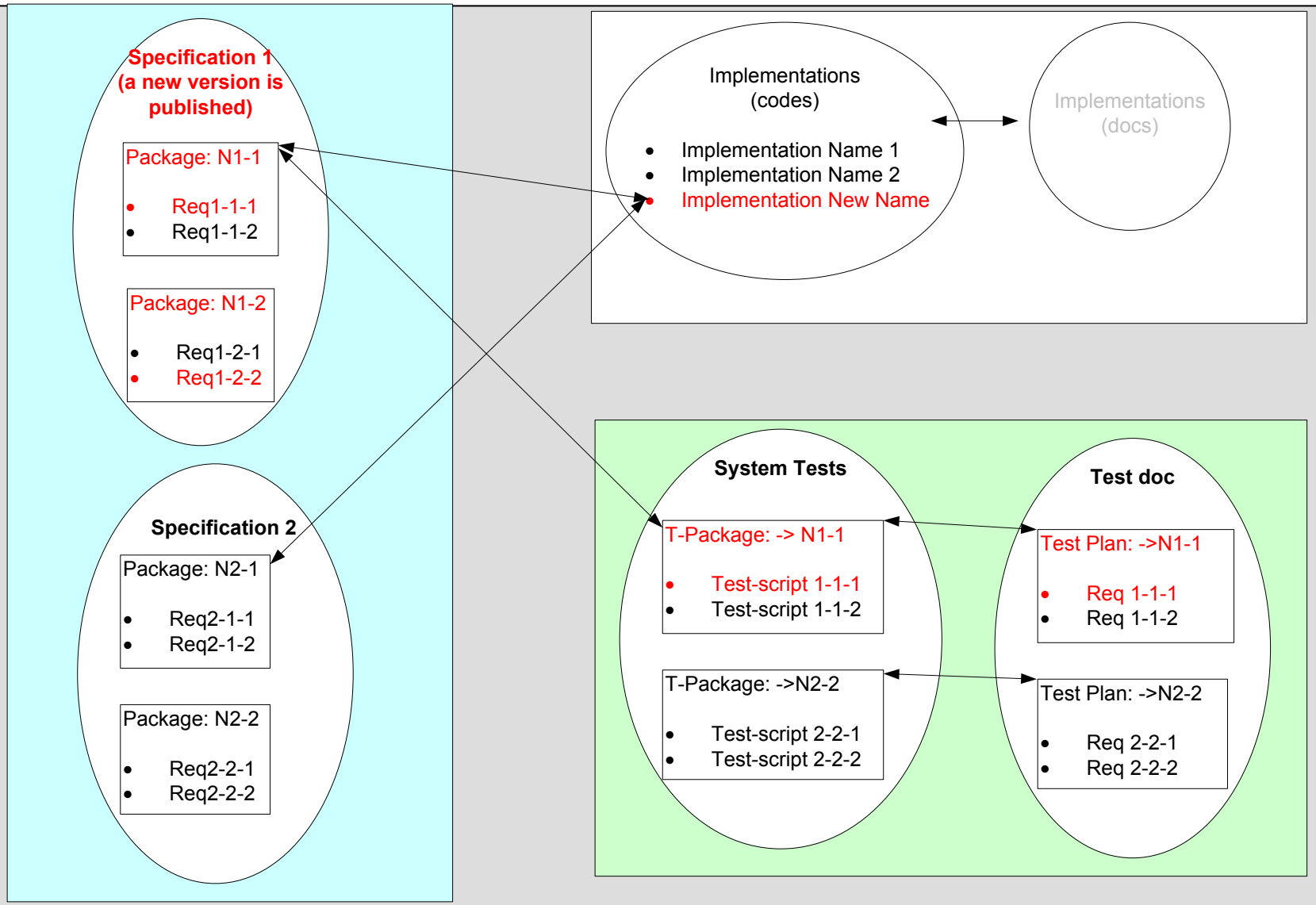
Dr. Amar Khelil

- **1** (3)Test Development; Situation and Requirements
- **2** (6)Test Development; Standard Procedure and Limitation
- **3** (2) Methodology proposed by LEIRIOS
- **4** (3) Pilot project with LEIRIOS (01.05-03.05)
- **5** (3) On-going projects with LEIRIOS
- **6** (2) Issues/Conclusions

1-1 Documents involved in Test Development



1-2 Test Development; one Use Case



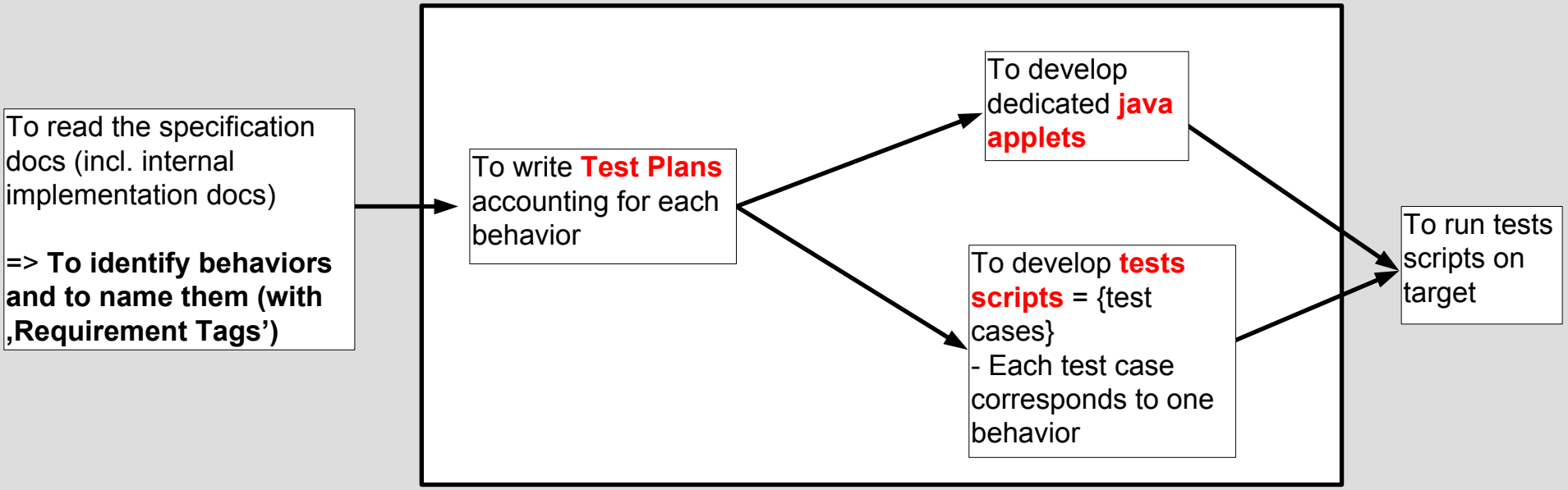
1-3 Requirements to Test Development

- Identify all relevant existing tests for **existing targets**
- Identify test development needs (new tests, modified existing tests) for **new targets**

For all targets

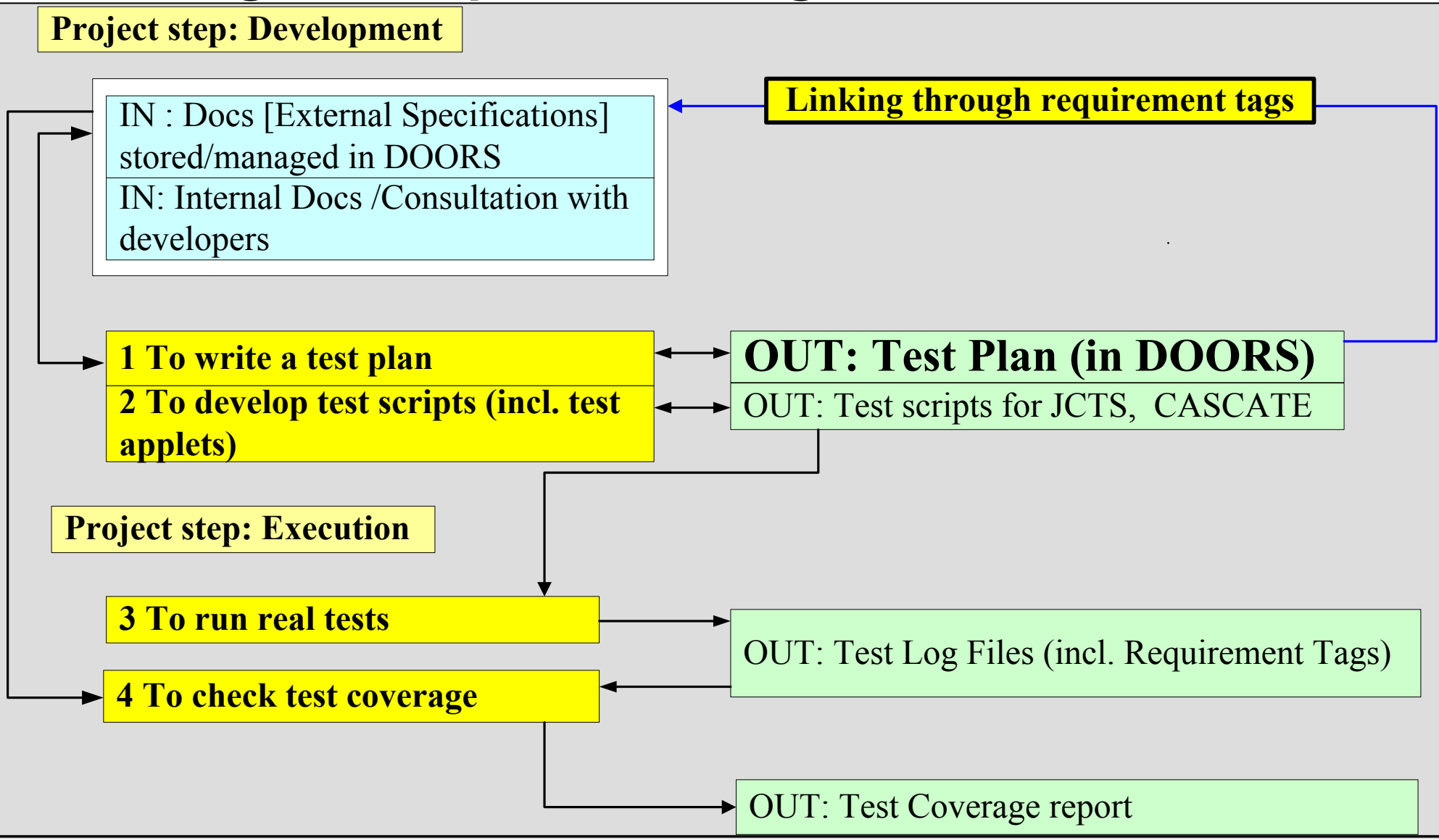
- **Reproducibility** of tests
- Measurement of **Test Coverage (?)**

2.1 Standard Procedure in place



Proof of **Reproducibility** and **Test Coverage** is done by tracing the ‚requirement tags' automatically (DOORS scripts)

2-2 Tracing the Requirement Tags



2.3 Test Plan

- is generated **manually**, as a **list of Test Case Definitions** (each one corresponding to a DOORS item)
- Correctness is assured by **Reviews**

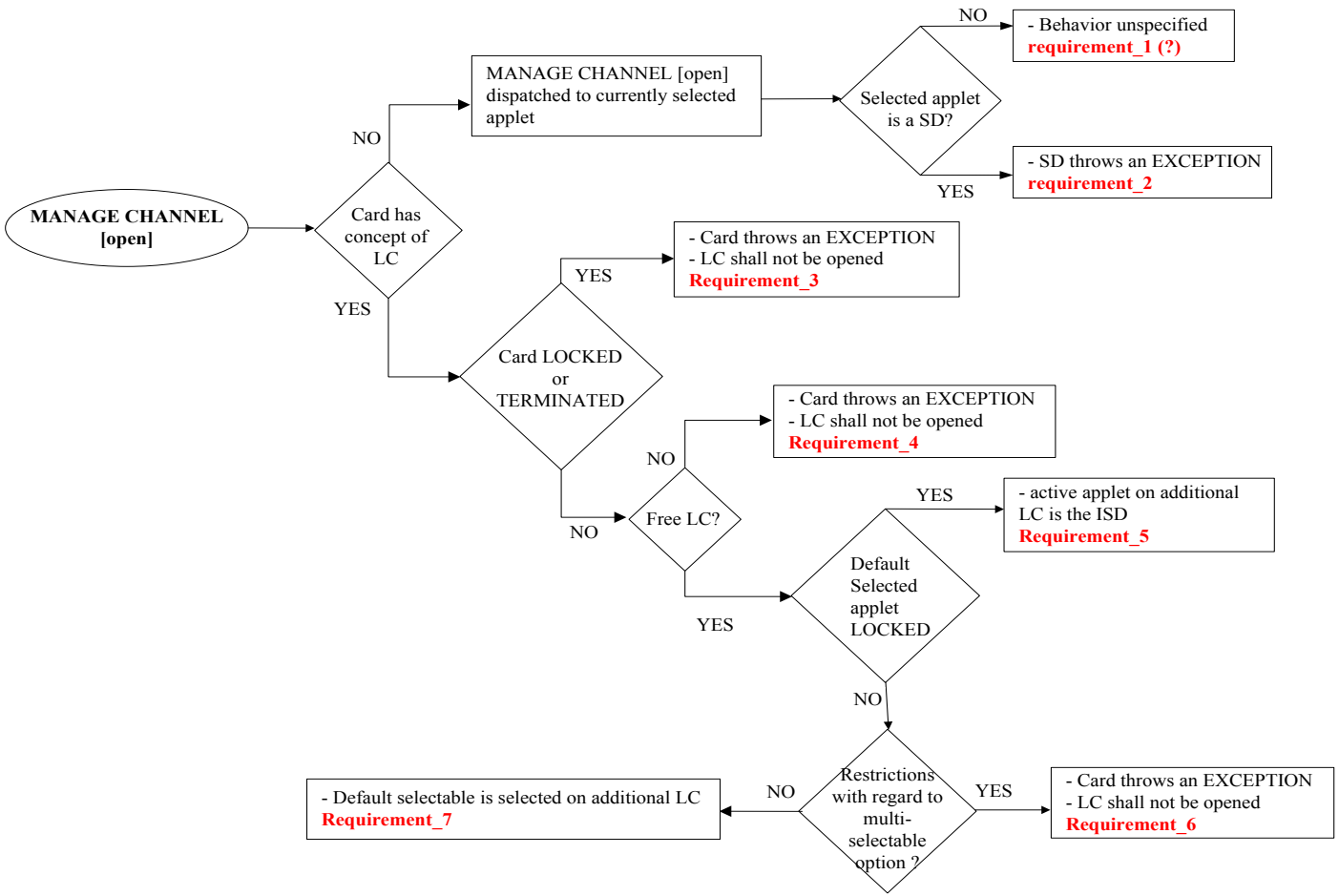
2.4 Test Plan

- is generated **manually**, as a **list of Test Case Definitions** (each one corresponding to a DOORS item)
- Correctness is assured by **Reviews**

Problem:

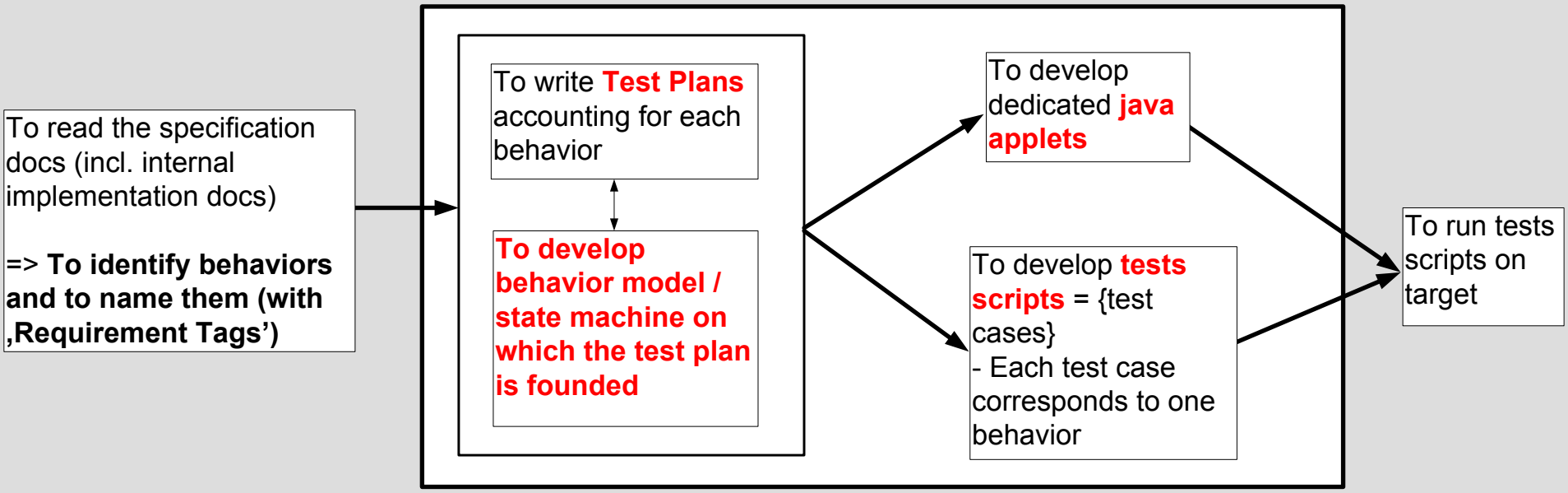
The Model of Behavior implicitly assumed by the Test Developer is not visible. Therefore, it is difficult to assess Test Coverage as regards content.

2.5 Example of model = Behavior(state of `machine`)



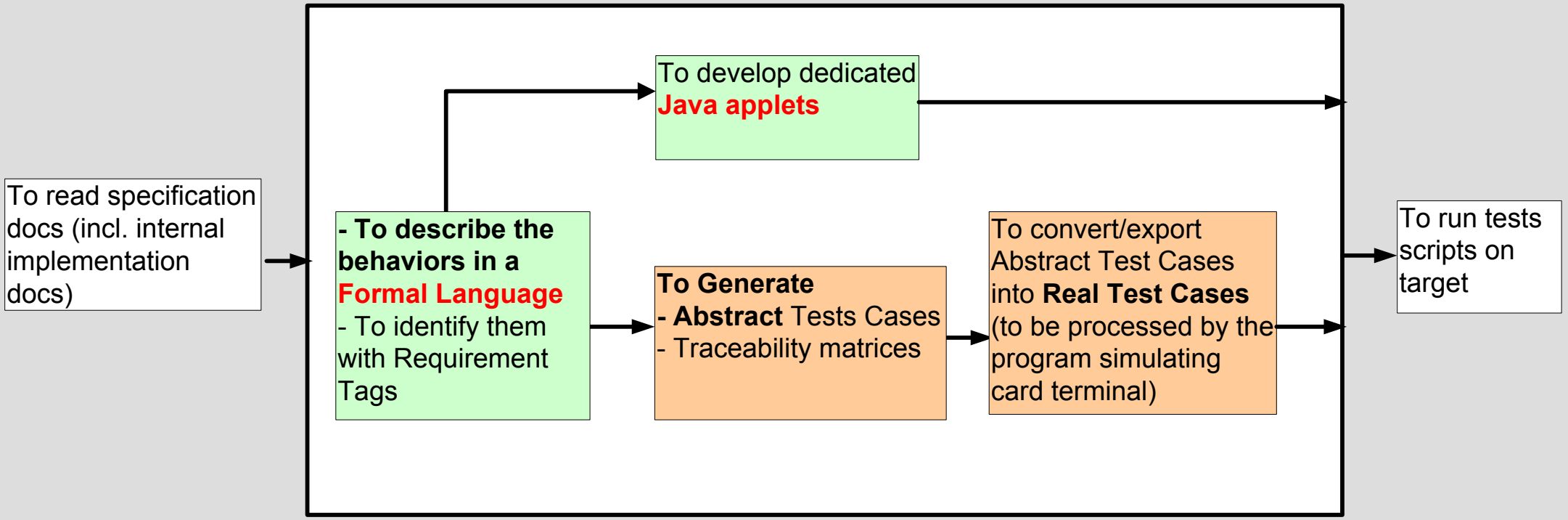
MANAGE CHANNEL [open] on basic LC; behavior described by GP2.1.1, JC2.2 not accounted for.

2.6 Test Plan Revisited



The Model of Behaviors is visible in Test Plan

3.1 Methodology proposed by LEIRIOS



3.2 LEIRIOS Test Generator (LTG)

- Formal Language used to identify behaviors is the **B-Method**
- Each Test Object is modelled as a **B-Machine** by means of Open-Source Editor **JEdit**, extended with **B-plugins**
- LTG provides an environment in order to define/manage Test Campaigns (GUI, Batch mode) and corresponding **Abstract** Test Cases (TC)
- LTG stores all TC pertaining to a specific model into dedicated DB and provides **Treacibility Matrices** (Coverage Check)
- LTG provides an interface (accessible through Open-Source **groovy** Script Language) in order to **convert Abstract TC into Real TC** (i.e. G&D proprietary CASCATE-Format)

4.1 Pilot Project (LEIRIOS/G&D-3FE-24, 01.05-03.05)

Focus:

To assess feasibility of the LEIRIOS methodology on a limited part of GP:

- Handling of **Secure Channels** (SCP01-simplified)
- Handling of **Logical Channels** (very simplified)

Output:

- A **syntactical** B-model + a set of Abstract tests
- A **functional** B-model + a set of Abstracts tests
- final report by G&D (in German)

Version of LTG: 2.0

4.2 Pilot Project with LTG: Syntactic Model

	Item	Number	Remarks
1	**Set**	7	Static part of the model description
2	**Constants**	27	Static part of the model description
3	**State variables**	0	Dynamic part of the model description - The behavior of machine does not depend on its states
4	**Operations**	6	Dynamic part of the model description - each operation represents an APDU command or part of it **1. UNKNOWN_APDU** **2. SELECT_APDU** **3. MANAGE_CHANNEL_APDU** **4. INITIALIZE_UPDATE_APDU** **5. EXTERNAL_AUTHENTICATE_APDU** **6. SET_STATUS_APDU_CARD** - in all cases the input parameters of operations refer to the APDU header
5	**Test campaigns**	2	- `AllLogicalChannels` (**19** test cases generated) - `BasicLogicalChannel` (**76** test cases generated)
6	**Missing tests**	>0	- Due to deficiency of the syntax model (no state variables accounted for) - Due to specific values of the test generation parameters - Due to the static model description (sets), there are non reachable test cases

Tab. Synopsis of the **Syntactic Model** (**314 lines of code**, 408 lines of code + comments)

4.3 Pilot Project with LTG: Functional Model

	Item	Number	Remarks
1	**Set**	8	Static part of the model description
2	**Constants**	3	Static part of the model description - All defined within the DEFINITION clause
3	**State variables**	20	Dynamic part of the model description
4	**Operations**	8	Dynamic part of the model description - each operation represents an APDU command or part of it **1. MANAGE_CHANELL_APDU_OPEN** **2. MANAGE_CHANELL_APDU_CLOSE** **3. SELECT_APDU_FIRST_BY_COMPLETE_NAME** **4. SET_STATUS_APDU_CARD** **5. INITIALIZE_UPDATE_APDU_SCP01** **6. INITIALIZE_UPDATE_APDU_SCP02** **7. EXTERNAL_AUTHENTICATE_APDU** 8. **GET_STATUS_APDU_ISD_TABLE922** - in all cases the input parameters of operations refer to the APDU header
5	**Test campaigns**	4	- `CardLifCycleState` (**26** test cases generated) - `ManageChannel` (**11** test cases generated) - `Select` (**13** test cases generated) - `SM_SCP_01` (**19** test cases generated)
6	**Missing tests**	?	Analysis is performed within GP2.1.1

Tab. Synopsis of the **Functional Model** (**862 lines of code**, 1056 lines of code + comments)

5.1 On-going Projects with LEIRIOS 04-11.2005

Focus: Development of (new) tests for the following packages

- **Secure Channels SCP01/SCP02** (M. Uminska, 3FE-22)
- **Life Cycle State Machine** (ISD, SD, Applets) (dropped)
- **Dispatcher** (A. Khelil, 3FE-24)

5.2 Overview of Dispatcher project -1-

The Dispatcher model/machine identifies

- **all** behaviors specified in **JC2.2** and **GP2.1.1**
- additional **unspecified** behaviors detected by simply attempting to **logically close the model**
  => requires input by the implementers
- behavioral **contradictions** between **JC2.2** and **GP2.1.1** (1)

The Dispatcher model/machine does **not** explicitly accounts for

- Secure Messaging

Conversion into **Real Test Cases** not yet started

5.3 Overview of Dispatcher project -2-

	Item	Number	Remarks
1	**Set**	14	Static part of model description
2	**Constants**	32	Static part of model description
3	**State variables**	26	Dynamic part of model description
4	**Operations**	8	Dynamic part of the model description - each operation in test focus represents at least one APDU command **1. RESET_procedure (test focus)** **2. APDU_SELECT_byName (test focus)** **3. APDU_MANAGE_CHANNEL_open (test focus)** **4. APDU_MANAGE_CHANNEL_close (test focus)** **5. COMMAND_2_DISPATCH_NO_SM_NO_CDATA (test focus)** 6. TRANSITION_LCS_OF_CLIENT (preamble/postamble) 7. TRANSITION_LCS_OF_SD (preamble/postamble) 8. TRANSITION_LCS_OF_CARD (preamble/postamble)
5	**Behavior switches**	10	
6	**Test campaigns**	-	On-going
7	**Missing tests (not in model)**	-	Not yet analyzed

Tab. Synopsis of the **Dispatcher Model** (**3876 lines of code**, 5770 lines of code + comments)

6.1 Issues

Model Development and Animation

- For first realistic projects, support by B (and Tool) experts is necessary

Test Generation

- Different ways to write a behavior impact on the Automated Test Case Generation (the tool may or may not find preambles)

Conversion into Real Test Cases

- The Test Script Language must support a level of abstraction compatible with the LTG output

LTG-Tool (versions 2->2.1.1)

- Bugs and/or cryptic error messages (LEIRIOS reacts quickly in those cases)
- LTG 2-2.1.1. does not fit completely into the G&D Test Development Process
- Documentation to LTG-DB Interface (for conversion into Real Test Case) is missing
- Version 2.2 (End of 2005) will address/solve most pending issues

6.2 Conclusions

- Formal Modeling of Specifications improves significantly the Test Development Process
- If LTG-2.2 keeps its promise, there are good chances that B Modeling, in conjunction with LTG, will be adopted as approved Test Development Procedure by G&D
- Are there alternative experiments with Formal Modeling of Specifications targeting the Test Generation Process out there ?

THANK YOU FOR ATTENTION

B Method

•The **B Method** was developed in order to automatically verify the consistency of logical structures (e.g. programs) , whereas consistency means that static and dynamic parts of model fits together

•**Therefore B** allows description of an object including :

- **Static Description**: **What is this object made of?**
  => includes *sets*, *constants*, *invariants*
- **Dynamic Description: How does it behaves?**
  => includes *variables*, *initialisation* state, and all possible state transition (*operations*)

Example of **B model (part I)** [see B-method, S. Schneider]

Nr.	**Klausel**	Eigenschaft	Beispiele / Kommentar
1	**MACHINE**	obligatorisch	`MACHINE Hotelguests(sze)` - `sze` number of rooms in hotel - ltg (LEIRIOS Tool) unterstützt keine Parameter
2	**CONSTRAINT**	optional	`CONSTRAINTS sze ∈ N` - Da sich die Klausel nur auf Parameter beziehen kann, wird sie auch nicht von ltg unterstützt.
3	**SETS**	optional	`ROOM; NAME; REPORT = {present, absent}` - Sätze werden Großgeschrieben - ltg unterstützt nur definierte finite SÄTZE
4	**CONSTANTS**	optional	`empty` - Variablen werden kleingeschrieben.
5	**PROPERTIES**	optional	`Card(ROOM) = sze ∧ empty ∈ NAME` - Es besteht die Möglichkeit in diesem Abschnitt das Äquivalent eines C-Makro zu definieren

Example of a **B model (part II)** [see B-method, S. Schneider]

Nr.	**Klausel**	Eigenschaft	Beispiele / Kommentar
6	**VARIABLES**	obligatorisch	`guests`
7	**INVARIANT**	obligatorisch	`guests ∈ ROOM → NAME`
8	**INITIALISATION**	obligatorisch	`guests := ROOM x {empty}`

9	**OPERATIONS**	obligatorisch	`guestcheckin(rr,nn) =`
			`PRE rr ∈ ROOM ∧ nn ∈ NAME ∧ nn ≠ empty`
			`THEN guests(rr) := nn`
			`END;`
			`guestcheckout(rr) =`
			`PRE rr ∈ ROOM`
			`THEN guests(rr) := empty`
			`END;`
			`nn ← guestcheckquery(rr) =`
			`PRE rr ∈ ROOM`
			`THEN nn := guests(rr)`
			`END;`
			- Jede einzelne Operation beschreibt einen ‚atomaren' Übergang des Maschinenzustandes.
			- Innerhalb einer Operation kennzeichnet das Schlüsselwort PRE Vorbedingungen, die gelten müssen, damit der Übergang durchführbar ist.
			- Das Schlüsselwort THEN kennzeichnet die Beschreibung der Zustandsänderung (welche Zustandsvariablen werden wie aktualisiert)
			- Das Schlüsselwort `END` beendet die Definition einer Operation

10	**END**	obligatorisch	Schließt die Maschinenbeschreibung.

LTG: Defining a Test campaign -1-

No.	Item	Remarks
1	selection	- defines , which part of the B model must be accounted for in the test case generation: 1. Which Operations are included? 2. Which state variables?
2	coverage	- For each single operation (of the B Model to test), the coverage of behavior is controlled by two values: Value 1: 1. `[decision coverage]`, 2. `[decision/condition coverage]`, 3. `[modified condition/decision coverage]`, 4. `[multiple condition coverage]`, Value 2: 1. `[with distribution]` 2. `[without distribution]`

LTG: Defining a Test campaign -2-

No.	Item	Remarks
3	Equivalent boundary values	- For each operation to test, possible option values are: 1. [one value], 2. [several values], 3. [all values]
4	Pairs of behavior coverage	- For each operation to test, the user can define operations that must follow the actual test case to be included. Following options are available: 1. [All pairs]: all accessible behaviors are included in the test case output 2. [Related pairs]: only behaviors are selected, that manipulate state variables, that were also manipulated in the actual test case 3. [Effect-cause]: only behaviors are selected, that manipulate state variables, whose values have been changed by the actual test case
5	preamble	The user can enter additional data to control the calculation of preambles. 1. Whether preambles be calculated or not? 2. maximal number of operations in one preamble 3. maximal duration of calculation for one preamble 4. Search algorithm : [width / depth] 5. Search algorithm : [backward/forward] 6. Filter of operations (that can possibly be included) 7. Manual input of a preambles
6	postamble	- Like preambles LTG can be triggered to calculate postambles, performing a return to the initial state of execution after a test case has been checked