Authoring Domain Specific Languages in Spreadsheets Using XML Technologies

Alan Painter
Development Engineer
HSBC France

HSBC

XML Prague
8 February 2019

Domain Specific Languages – The Short Description

What is a DSL?

- a computer language specialized to a particular application domain

make

```
all: hello.exe

hello.exe: hello.o

hello.o

hello.o: 


clean:
           rm hello.o
hello.exe
```

YACC

```
statement_list
       : statement
       | statement statement_list
```

```
       | expression TIMES expression {$$ = $1 * $3;}
       | expression DIV   expression {$$ = $1 / $3;}
       | MINUS expression %prec UMINUS {$$ = - $2;}
       | '(' expression ')' { $$ = $2; }
       | NUMBER
       | NAME   { $$ = vbltable[$1]; }
```

Two main aspects of any DSL:

- The syntax of the language
- The implementation of the action (often generating an artifact)

troff

```
.nf
.ll 4.0i
.in 2.0i
101 Main Street
Morristown, NJ  07960
15 March, 1997
.sp 1i
.in 0
Dear Sir,
```

html

```
<p>You can reach Michael at:</p>
<ul>
  <li><a href="https://example.com">Website</a></li>
  <li><a href="mailto:m.bluth@example.com">Email</a></li>
  <li><a href="tel:+123456789">Phone</a></li>
</ul>
```

Domain Specific Languages – Diverse Uses

In a paper presented October 2018 to the ACM/IEEE Conference *MODELS '18*, Juha-Pekka Tolvanen and Steven Kelly presented a survey of DSLs and the effort required to develop them. The DSLs surveyed were in diverse domains:

- Voice control systems for home automation
- Testing a military radio system
- Touch screen controller

Their survey noted that the it required from a few person-days to 3 person-weeks to develop the DSLs.

Domain Specific Languages – What’s the Utility?

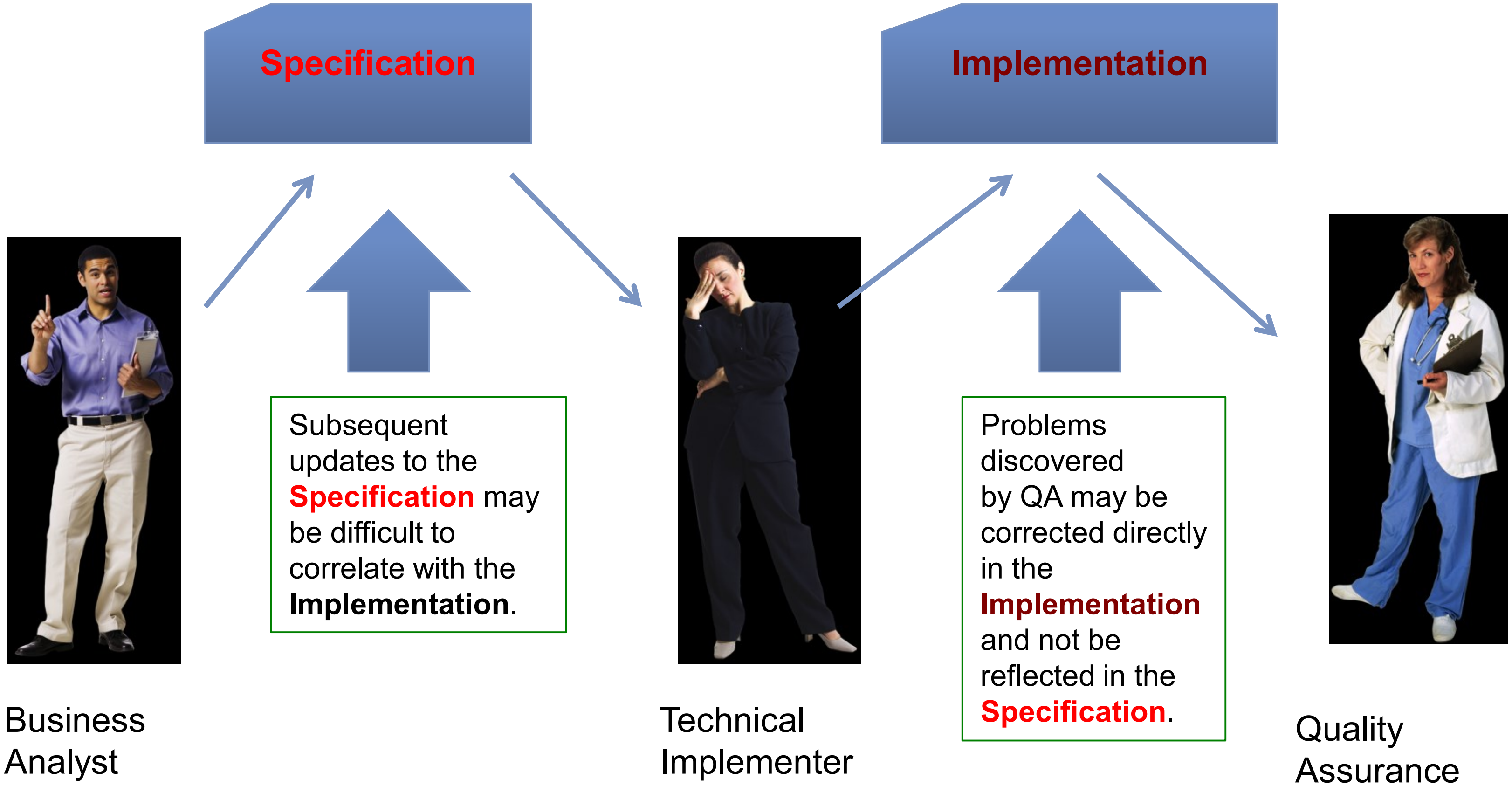
« I believe that the hardest part of software projects, the most common source of project failure, is communication with the customers and users of that software. By providing a clear yet precise language to deal with domains, a DSL can help improve this communication. »

—Martin Fowler, *Domain Specific Languages*, 2010

«[XML] is terrible for a programming language. Once you start putting structures like control logic the noise of XML becomes intolerable. The great example of this is XSLT, which is awful to work with. No language can be good that makes a subroutine call so painful. »

—Martin Fowler, *Use of XML*, 3 January 2014

A Typical Development Process Without a DSL



Specification

Implementation

Subsequent updates to the **Specification** may be difficult to correlate with the **Implementation**.

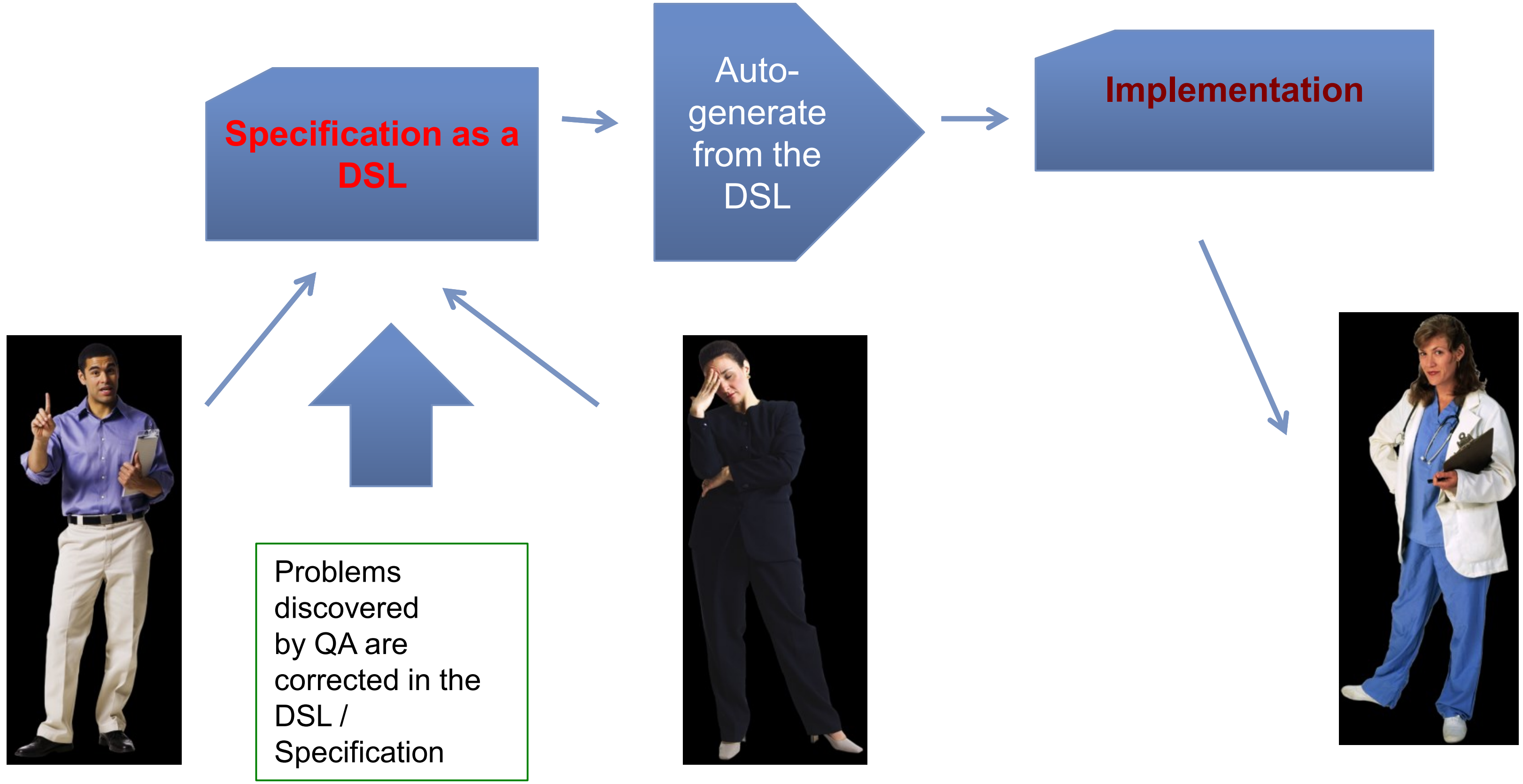
Problems discovered by QA may be corrected directly in the **Implementation** and not be reflected in the **Specification**.

Business Analyst

Technical Implementer

Quality Assurance

A Cleaner Development Process With a DSL



Specification as a DSL

Auto-generate from the DSL

Implementation

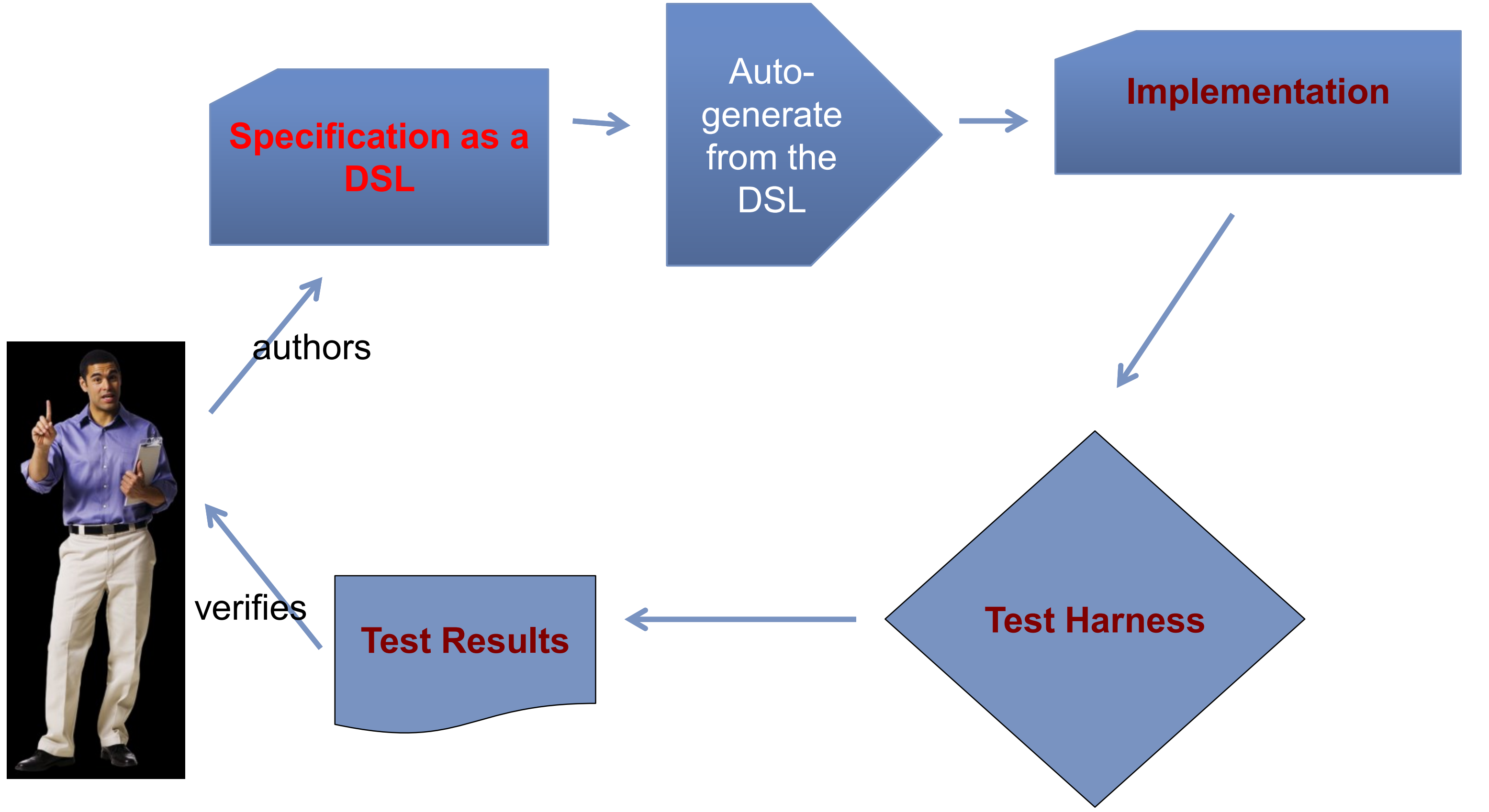
Problems discovered by QA are corrected in the DSL / Specification

Business Analyst

Technical Implementer

Quality Assurance

A Shorter Testing Cycle With a DSL



Specification as a DSL

Auto-generate from the DSL

Implementation

Test Harness

Test Results

authors

verifies

Business
Analyst

DSLs in Spreadsheets

- Business Analysts and Domain Experts are Extremely Comfortable working with Spreadsheets
- Everything is squared up.
  - Rows can be lined up for tablular data → readability
- The editing model allows for editing blocks of cells, entire rows, entire columns.
- Even multi-line text can be contained within a cell
- Can add text colors, styles, background colors, etc *(i.e. Pimp my spec!!!)*

Summing up: The Value Proposition

If we accept that :

- DSLs present a meaningful and readable expression of a process
- Business Analysts can use DSLs to be direct contributors to development
- Business Analysts prefer to work with spreadsheets

We should use spreadsheets as a support for DSLs!

But wait, there’s more!

- XML Technologies (Xquery, XSLT) can read spreadsheets easily
- XML Technologies (Xquery, XSLT) can produce almost any artifact

We can use XML Technologies for implementing DSLs in Spreadsheets.

Why is it that XML Technologies can read a spreadsheet document so easily?

Spreadsheet documents are already XML!

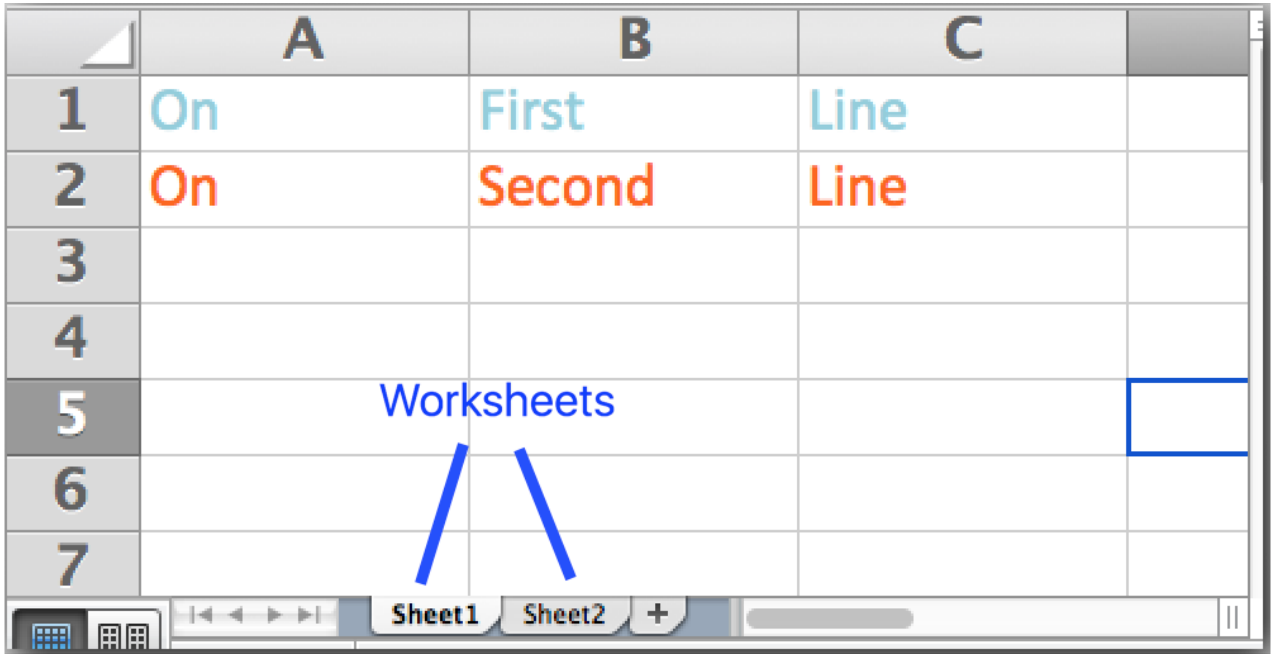
- Microsoft XML Format (**.xml**)
    - a single xml Document (Office 2003)

- Open Office XML (OOXML) (**.xlsx**)
    - a zip archive containing a collection of XML files
    - 2 major versions of the XML content

- Open Document Format (ODF) (**.odt**)
    - a zip archive containing a collection of XML files

The Simple Model for Data in a Spreadsheet

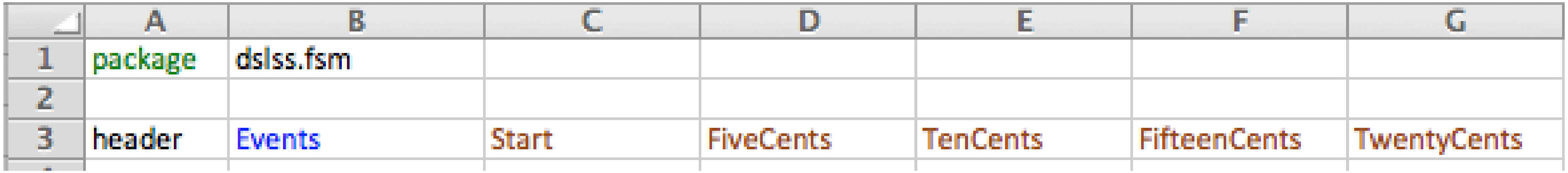
```
<Workbook>
   <Worksheet name="Sheet1">
      <Line>
         <Cell>On</Cell>
         <Cell>First</Cell>
         <Cell>Line</Cell>
      </Line>
      <Line>
         <Cell>On</Cell>
         <Cell>Second</Cell>
         <Cell>Line</Cell>
      </Line>
   </Worksheet>
   <Worksheet name="Sheet2">
      ....
   </Worksheet>
</Workbook>
```



	A	B	C
1	On	First	Line
2	On	Second	Line
3			
4			
5			
6			
7			

Worksheets

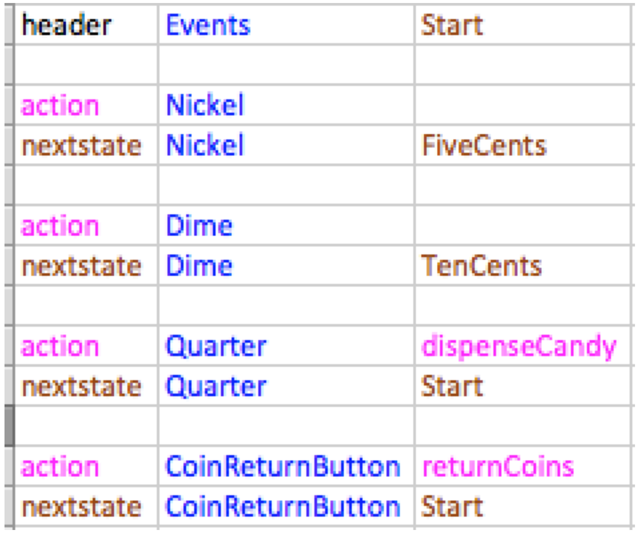
Using our Simple Model to Design a Spreadsheet DSL



	A	B	C	D	E	F	G
1	package	dslss.fsm					
2							
3	header	Events	Start	FiveCents	TenCents	FifteenCents	TwentyCents

```
<xsl:function name="f:getPackage" as="xs:string">
    <xsl:param name="lines" as="element(Line)*" />
    <xsl:sequence select="$lines[Cell[1] eq 'package' ]/Cell[2]" />
</xsl:function>

<xsl:function name="f:getStates" as="xs:string*" >
    <xsl:param name="lines" as="element(Line)*" />
    <xsl:sequence select="$lines[Cell[1] eq 'header']/Cell[position() gt 2]" />
</xsl:function>

<xsl:function name="f:getHeaderIndex" as="xs:integer" >
    <xsl:param name="lines" as="element(Line)*" />
    <xsl:param name="state" as="xs:string"      />

    <xsl:variable name="headerCells" select="$lines[Cell[1] eq 'header']/Cell"
                  as="xs:string*" />
    <xsl:sequence select="index-of($headerCells, $state)" />
</xsl:function>
```

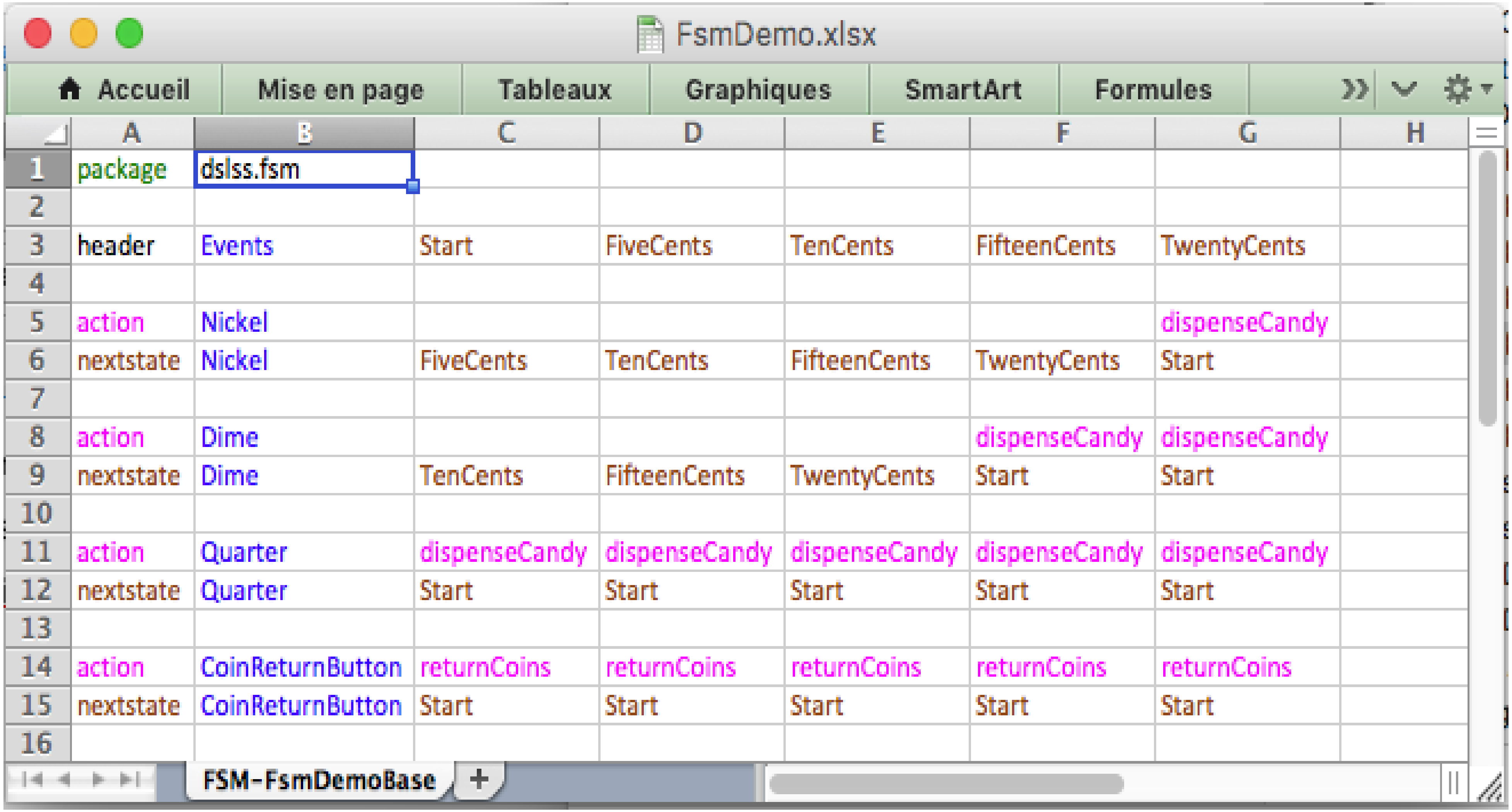
Using our Simple Model to Design a Spreadsheet DSL(2)



header	Events	Start
action	Nickel	
nextstate	Nickel	FiveCents
action	Dime	
nextstate	Dime	TenCents
action	Quarter	dispenseCandy
nextstate	Quarter	Start
action	CoinReturnButton	returnCoins
nextstate	CoinReturnButton	Start

```
<xsl:function name="f:getEvents" as="xs:string*" >
    <xsl:param name="lines" as="element(Line)*" />
    <xsl:sequence select="$lines[Cell[1] eq 'action']/Cell[2]" />
</xsl:function>

 <xsl:function name="f:getAction" as="xs:string?" >
     <xsl:param name="lines" as="element(Line)*" />
     <xsl:param name="state" as="xs:string"      />
     <xsl:param name="event" as="xs:string"      />

     <xsl:variable name="stateColumn" select="f:getHeaderIndex($lines, $state)"
                   as="xs:integer" />
     <xsl:sequence select="$lines[Cell[1] eq 'action']
                                 [Cell[2] eq $event  ]/Cell[$stateColumn]" />
 </xsl:function>
```

DSL of an Automaton (Finite State Machine)



	A	B	C	D	E	F	G	H
1	package	dslss.fsm						
2								
3	header	Events	Start	FiveCents	TenCents	FifteenCents	TwentyCents	
4								
5	action	Nickel					dispenseCandy	
6	nextstate	Nickel	FiveCents	TenCents	FifteenCents	TwentyCents	Start	
7								
8	action	Dime				dispenseCandy	dispenseCandy	
9	nextstate	Dime	TenCents	FifteenCents	TwentyCents	Start	Start	
10								
11	action	Quarter	dispenseCandy	dispenseCandy	dispenseCandy	dispenseCandy	dispenseCandy	
12	nextstate	Quarter	Start	Start	Start	Start	Start	
13								
14	action	CoinReturnButton	returnCoins	returnCoins	returnCoins	returnCoins	returnCoins	
15	nextstate	CoinReturnButton	Start	Start	Start	Start	Start	
16								

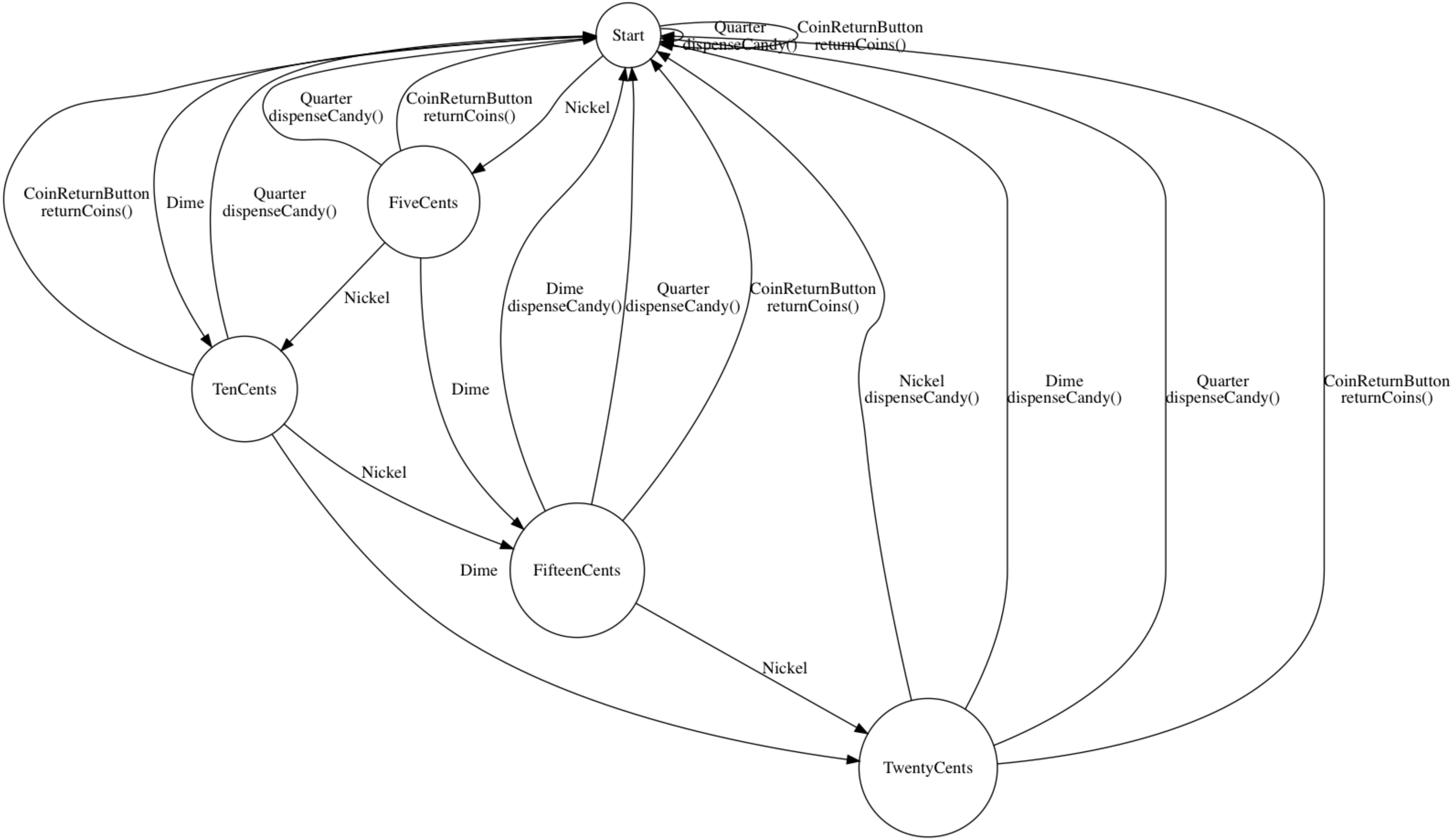
Generated Java Abstract Class

```
package dslss.fsm;

public abstract class FsmDemoBase {

    public enum Event { Nickel, Dime, Quarter, CoinReturnButton }

    public enum State { Start, FiveCents, TenCents, FifteenCents, TwentyCents }

    protected abstract Runnable dispenseCandy();
    protected abstract Runnable returnCoins();

    private final Runnable action[][] = {
        { __nop(),         __nop(),         __nop(),         __nop(),         dispenseCandy() },
        { __nop(),         __nop(),         __nop(),         dispenseCandy(), dispenseCandy() },
        { dispenseCandy(), dispenseCandy(), dispenseCandy(), dispenseCandy(), dispenseCandy() },
        { returnCoins(),   returnCoins(),   returnCoins(),   returnCoins(),   returnCoins()   },
    };

    private final static State nextState[][] = {
        { State.FiveCents, State.TenCents,     State.FifteenCents, State.TwentyCents, State.Start },
        { State.TenCents,  State.FifteenCents, State.TwentyCents,  State.Start,       State.Start },
        { State.Start,     State.Start,        State.TwentyCents,  State.Start,       State.Start },
        { State.Start,     State.Start,        State.TwentyCents,  State.Start,       State.Start },
    };

    public final State handleEvent(final State currentState, final Event newEvent) {
                  action [newEvent.ordinal()] [currentState.ordinal()].run();
        return nextState [newEvent.ordinal()] [currentState.ordinal()];
    }

    private Runnable __nop() { return () -> {}; }
}
```

Generated GraphViz

```
digraph FsmDemoBase {
    node [shape = circle];
    Start        -> FiveCents    [ label = "Nickel"                          ];
    Start        -> TenCents     [ label = "Dime"                            ];
    Start        -> Start        [ label = "Quarter\ndispenseCandy()"        ];
    Start        -> Start        [ label = "CoinReturnButton\nreturnCoins()" ];
    FiveCents    -> TenCents     [ label = "Nickel"                          ];
    FiveCents    -> FifteenCents [ label = "Dime"                            ];
    FiveCents    -> Start        [ label = "Quarter\ndispenseCandy()"        ];
    FiveCents    -> Start        [ label = "CoinReturnButton\nreturnCoins()" ];
    TenCents     -> FifteenCents [ label = "Nickel"                          ];
    TenCents     -> TwentyCents  [ label = "Dime"                            ];
    TenCents     -> Start        [ label = "Quarter\ndispenseCandy()"        ];
    TenCents     -> Start        [ label = "CoinReturnButton\nreturnCoins()" ];
    FifteenCents -> TwentyCents  [ label = "Nickel"                          ];
    FifteenCents -> Start        [ label = "Dime\ndispenseCandy()"           ];
    FifteenCents -> Start        [ label = "Quarter\ndispenseCandy()"        ];
    FifteenCents -> Start        [ label = "CoinReturnButton\nreturnCoins()" ];
    TwentyCents  -> Start        [ label = "Nickel\ndispenseCandy()"         ];
    TwentyCents  -> Start        [ label = "Dime\ndispenseCandy()"           ];
    TwentyCents  -> Start        [ label = "Quarter\ndispenseCandy()"        ];
    TwentyCents  -> Start        [ label = "CoinReturnButton\nreturnCoins()" ];
}
```

GraphViz Graphic



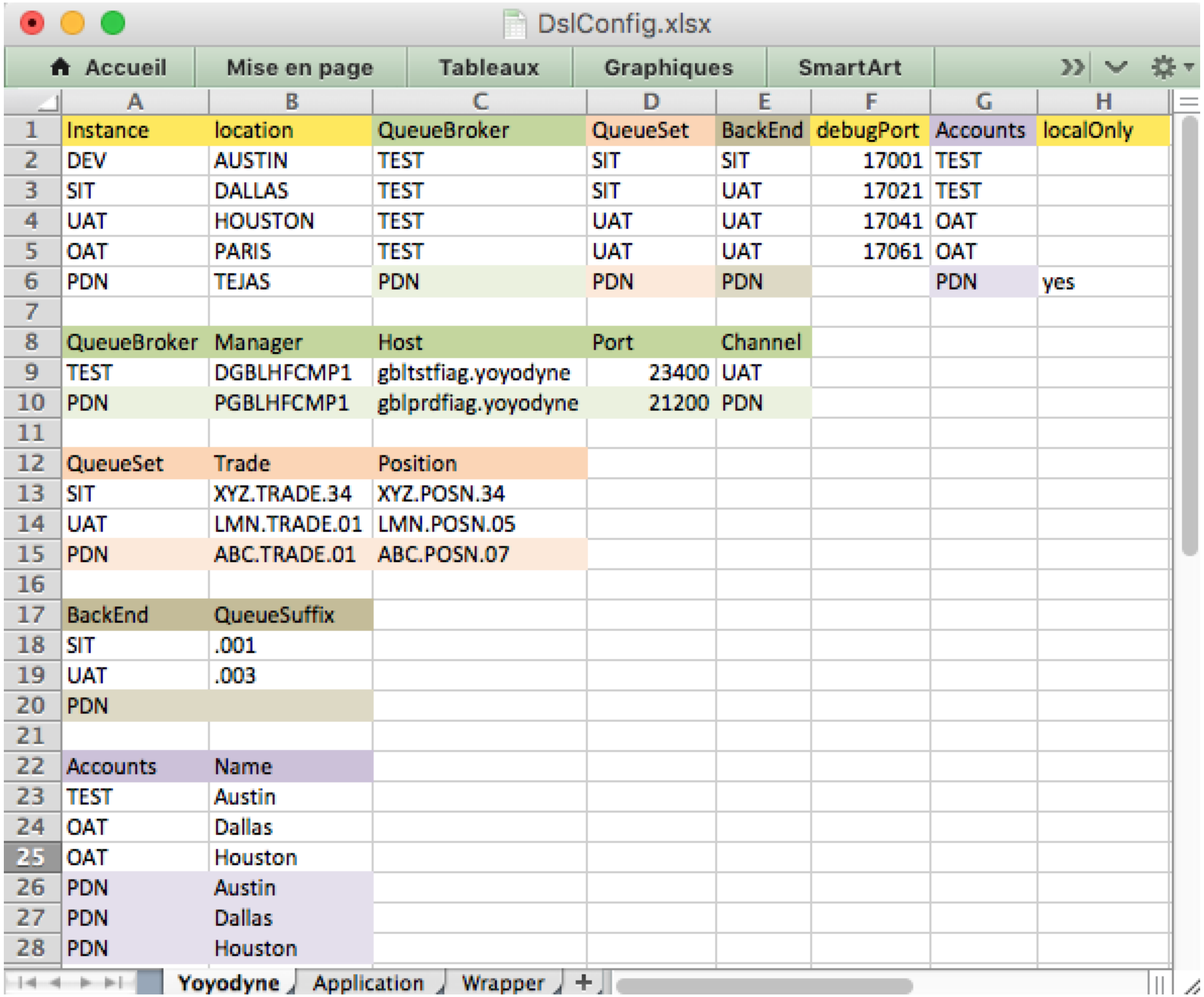
Generating an Application Configuration

- We have a large number of installed instances with different configurations.
- We want to have a central inventory of the instances and their different configurations.
- We'll generate at least some properties files (two in the example)

```
system.location=AUSTIN
jms.QUEUE_MGR=DGBLHFCMP1
jms.HOST_NAME=gbltstfiag.yoyodyne
jms.PORT=23400
...
```

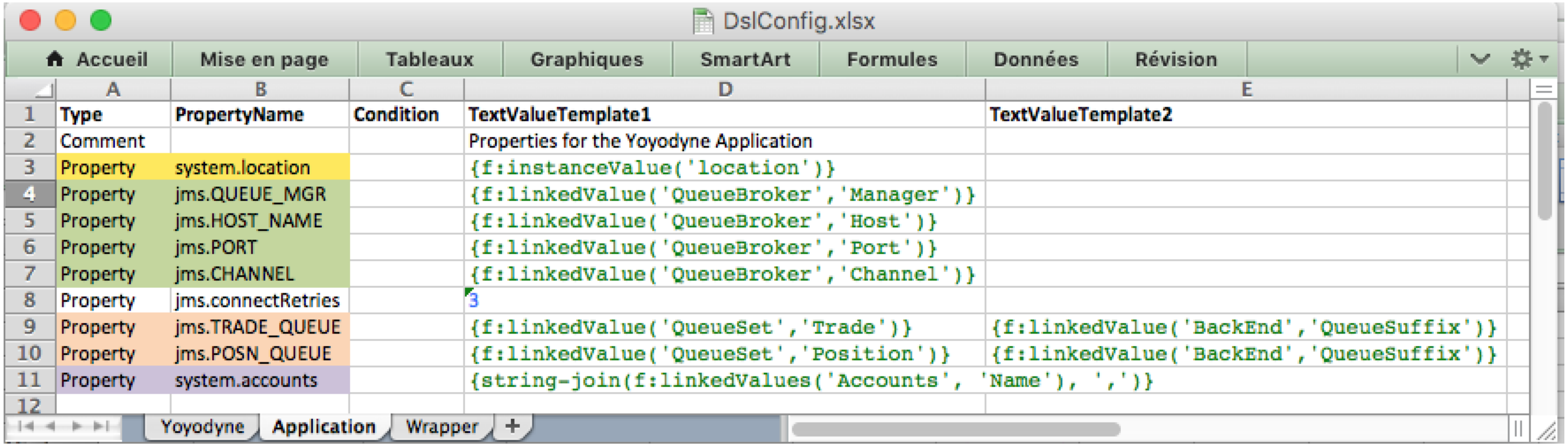
```
wrapper.java.additional.1=-Drmi.hostname=localhost
wrapper.java.additional.2=-Xms1024m
wrapper.java.additional.3=-Xmx1024m
wrapper.app.parameter.1=classpath:yoyodyne_service.xml
...
```

DSL Model for Generating an Application Configuration

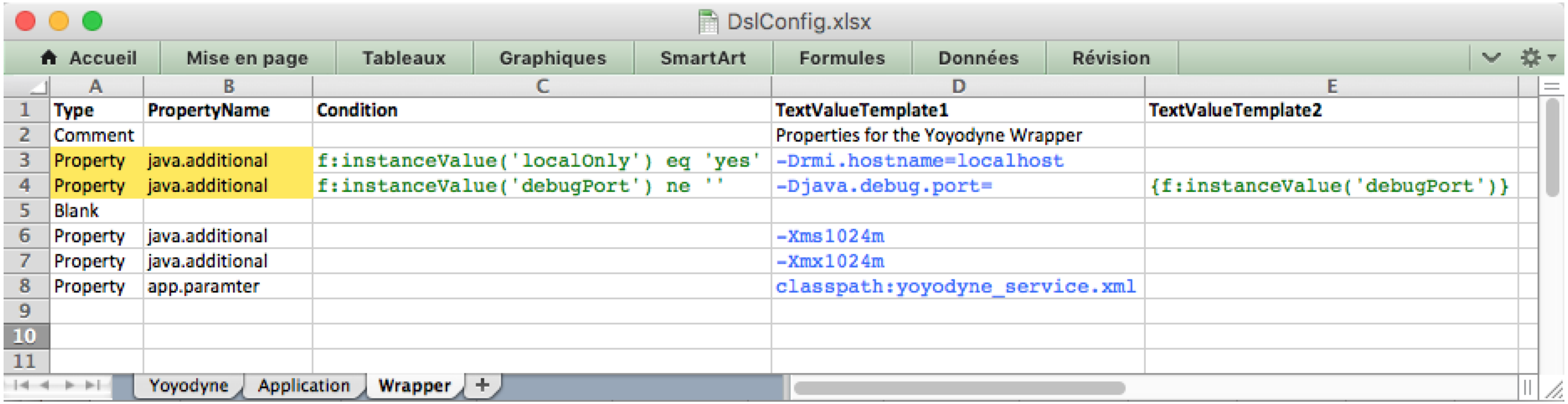


	A	B	C	D	E	F	G	H
1	Instance	location	QueueBroker	QueueSet	BackEnd	debugPort	Accounts	localOnly
2	DEV	AUSTIN	TEST	SIT	SIT	17001	TEST	
3	SIT	DALLAS	TEST	SIT	UAT	17021	TEST	
4	UAT	HOUSTON	TEST	UAT	UAT	17041	OAT	
5	OAT	PARIS	TEST	UAT	UAT	17061	OAT	
6	PDN	TEJAS	PDN	PDN	PDN		PDN	yes
7								
8	QueueBroker	Manager	Host	Port	Channel			
9	TEST	DGBLHFCMP1	gbltstfiag.yoyodyne	23400	UAT			
10	PDN	PGBLHFCMP1	gblprdfiag.yoyodyne	21200	PDN			
11								
12	QueueSet	Trade	Position					
13	SIT	XYZ.TRADE.34	XYZ.POSN.34					
14	UAT	LMN.TRADE.01	LMN.POSN.05					
15	PDN	ABC.TRADE.01	ABC.POSN.07					
16								
17	BackEnd	QueueSuffix						
18	SIT	.001						
19	UAT	.003						
20	PDN							
21								
22	Accounts	Name						
23	TEST	Austin						
24	OAT	Dallas						
25	OAT	Houston						
26	PDN	Austin						
27	PDN	Dallas						
28	PDN	Houston						

Templates for the Properties Files

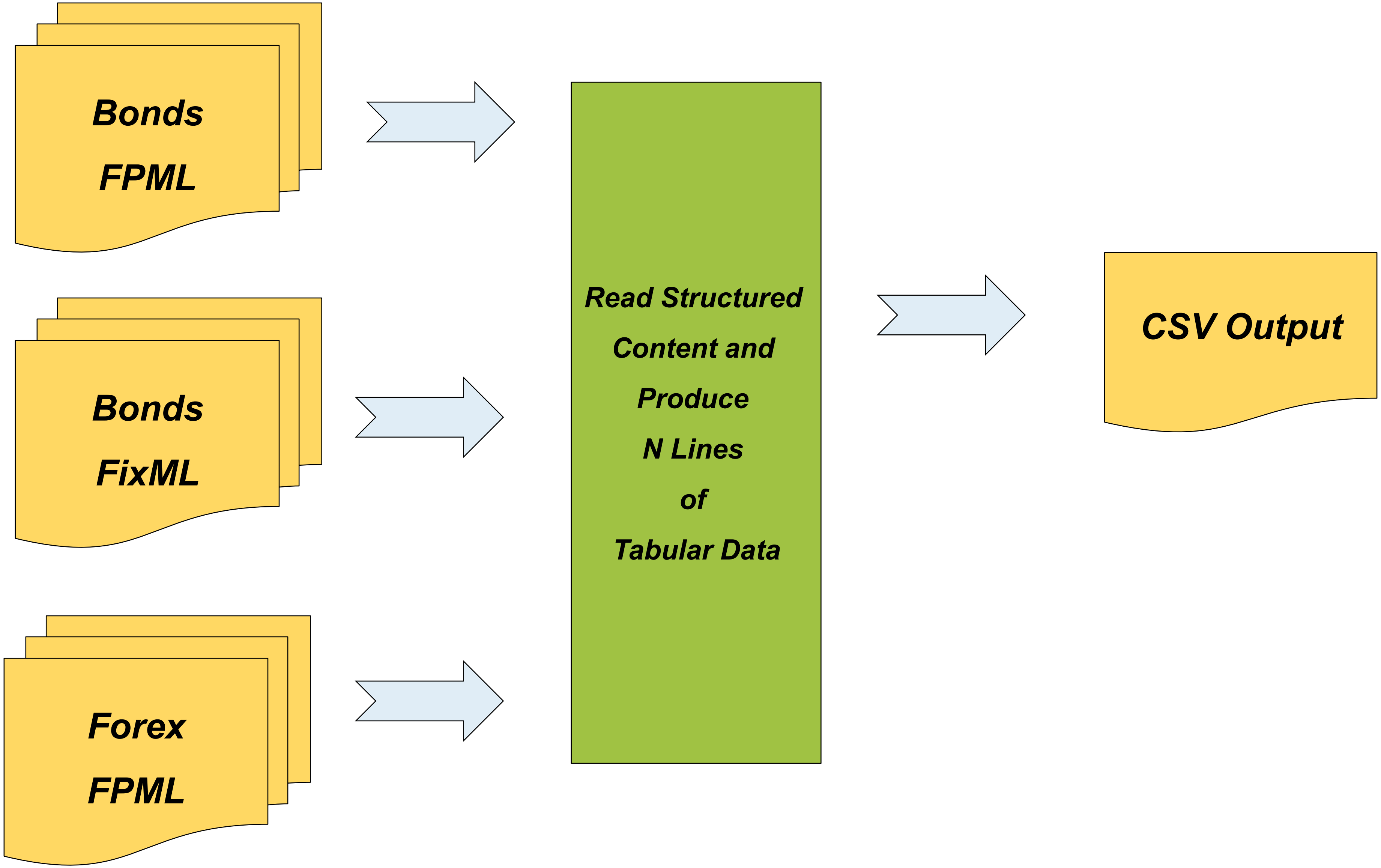


	A	B	C	D	E
1	Type	PropertyName	Condition	TextValueTemplate1	TextValueTemplate2
2	Comment			Properties for the Yoyodyne Application	
3	Property	system.location		{f:instanceValue('location')}	
4	Property	jms.QUEUE_MGR		{f:linkedValue('QueueBroker','Manager')}	
5	Property	jms.HOST_NAME		{f:linkedValue('QueueBroker','Host')}	
6	Property	jms.PORT		{f:linkedValue('QueueBroker','Port')}	
7	Property	jms.CHANNEL		{f:linkedValue('QueueBroker','Channel')}	
8	Property	jms.connectRetries		3	
9	Property	jms.TRADE_QUEUE		{f:linkedValue('QueueSet','Trade')}	{f:linkedValue('BackEnd','QueueSuffix')}
10	Property	jms.POSN_QUEUE		{f:linkedValue('QueueSet','Position')}	{f:linkedValue('BackEnd','QueueSuffix')}
11	Property	system.accounts		{string-join(f:linkedValues('Accounts', 'Name'), ',')}	
12					



	A	B	C	D	E
1	Type	PropertyName	Condition	TextValueTemplate1	TextValueTemplate2
2	Comment			Properties for the Yoyodyne Wrapper	
3	Property	java.additional	f:instanceValue('localOnly') eq 'yes'	-Drmi.hostname=localhost	
4	Property	java.additional	f:instanceValue('debugPort') ne ''	-Djava.debug.port=	{f:instanceValue('debugPort')}
5	Blank				
6	Property	java.additional		-Xms1024m	
7	Property	java.additional		-Xmx1024m	
8	Property	app.paramter		classpath:yoyodyne_service.xml	
9					
10					
11					

Extracting Tabular Data From Diverse Content Models



Bonds

FPML

Bonds

FixML

Forex

FPML

Read Structured

Content and

Produce

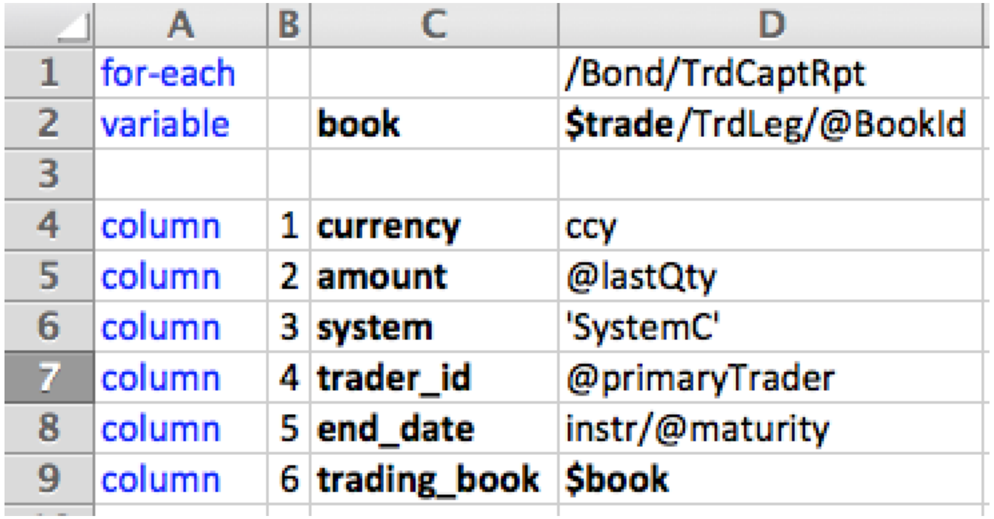
N Lines

of

Tabular Data

CSV Output

Primitive DSL

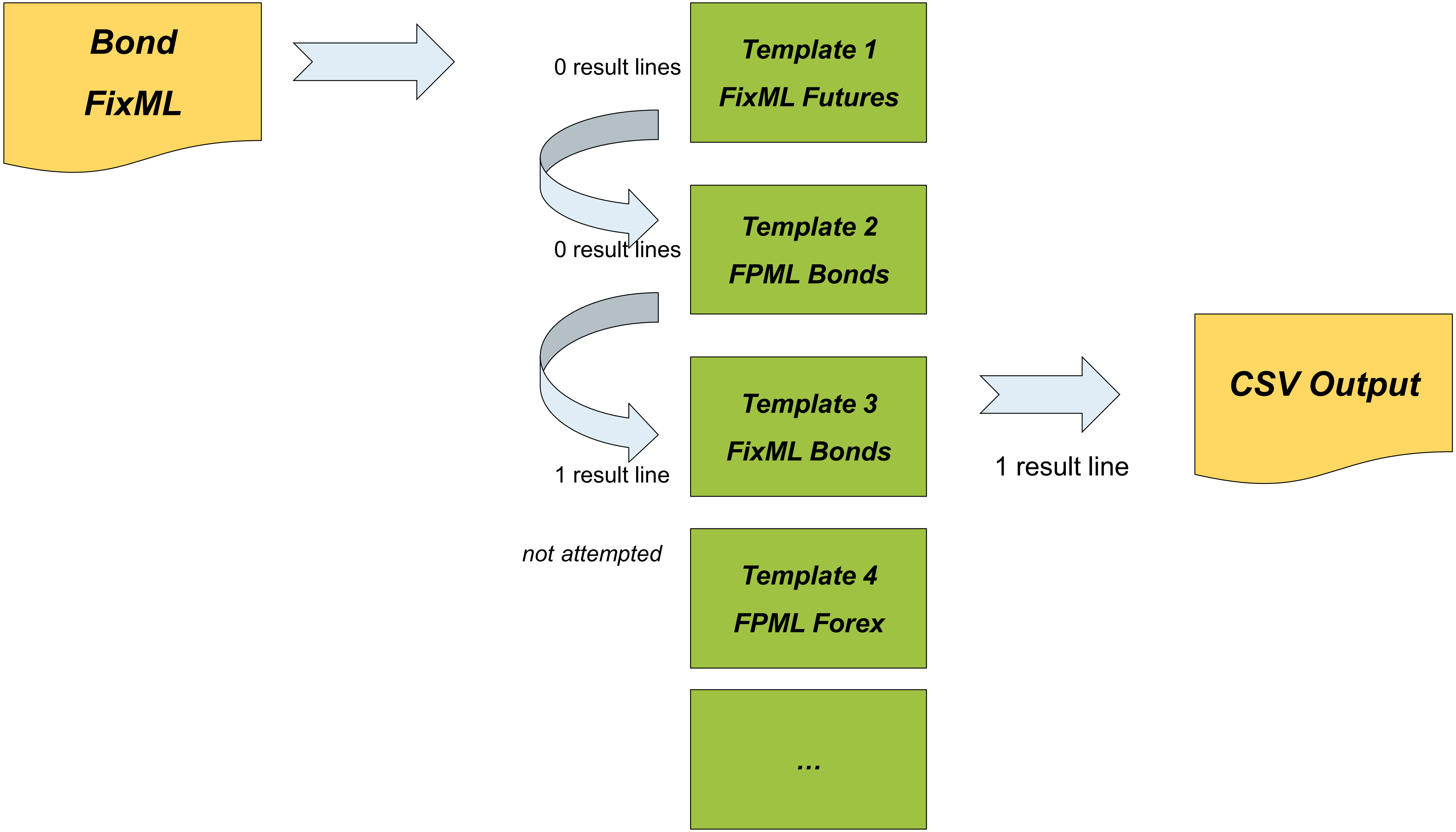


	A	B	C	D
1	for-each			/Bond/TrdCaptRpt
2	variable		**book**	**$trade**/TrdLeg/@BookId
3				
4	column	1	**currency**	ccy
5	column	2	**amount**	@lastQty
6	column	3	**system**	'SystemC'
7	column	4	**trader_id**	@primaryTrader
8	column	5	**end_date**	instr/@maturity
9	column	6	**trading_book**	**$book**

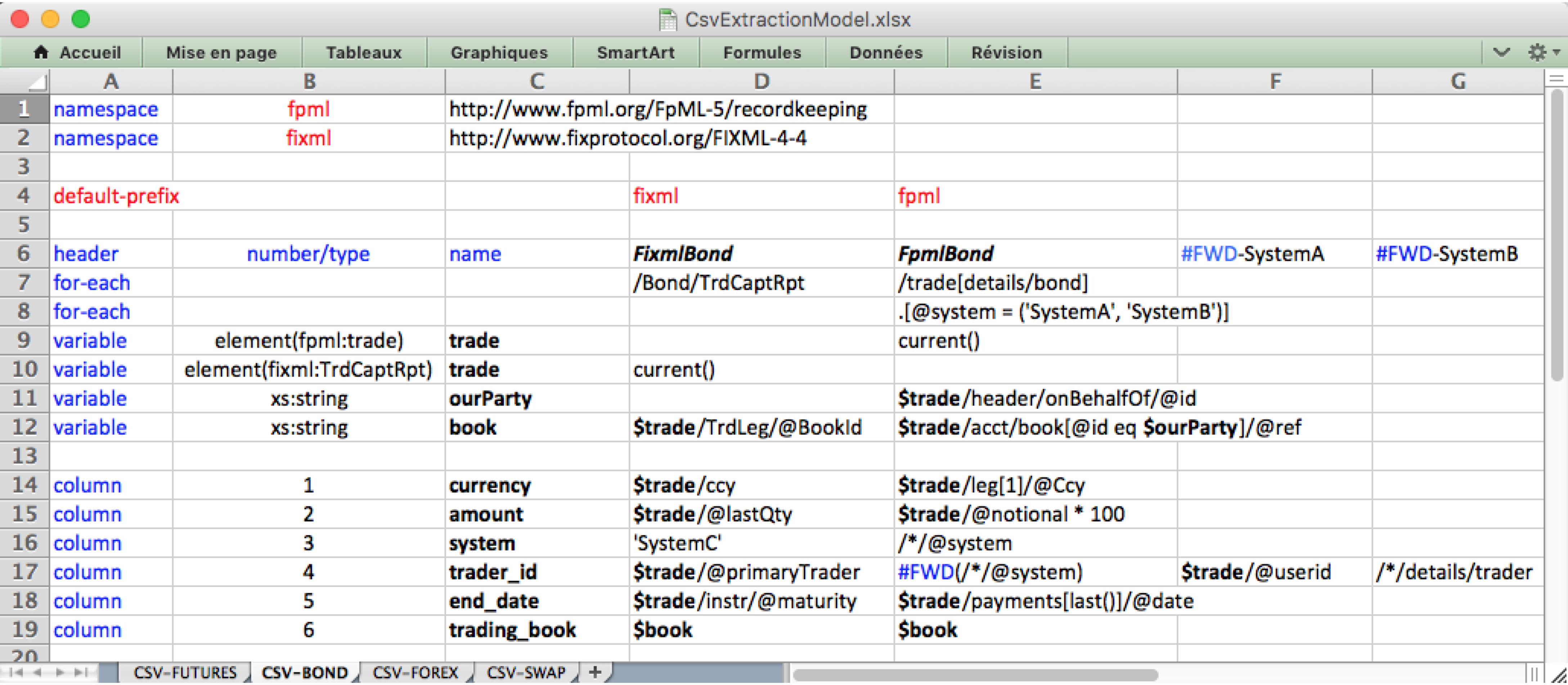
Generated XSLT Template From the Primitive DSL

```
<xsl:template xmlns:fpml="http://www.fpml.org/FpML-5/recordkeeping"
              xmlns:fixml="http://www.fixprotocol.org/FIXML-4-4"
              xpath-default-namespace="http://www.fixprotocol.org/FIXML-4-4"
              name="f:BOND-FixmlBond" as="xs:string*">
    <xsl:for-each select="/Bond/TrdCaptRpt">
        <xsl:variable name="book" as="xs:string" select="$trade/TrdLeg/@BookId" />
        <xsl:variable name="resultCells" as="item()*">
            <xsl:sequence select="f:empty-if-absent(ccy)" />
            <xsl:sequence select="f:empty-if-absent(@lastQty)" />
            <xsl:sequence select="f:empty-if-absent('SystemC')" />
            <xsl:sequence select="f:empty-if-absent(@primaryTrader)" />
            <xsl:sequence select="f:empty-if-absent(instr/@maturity)" />
            <xsl:sequence select="f:empty-if-absent($book)" />
        </xsl:variable>
        <xsl:value-of separator="{$separator}"
                      select="for $i in $resultCells
                              return f:encode-csv($i, $separator)" />
    </xsl:for-each>
</xsl:template>
```

Basic Mechanism for Choosing Rules to Apply

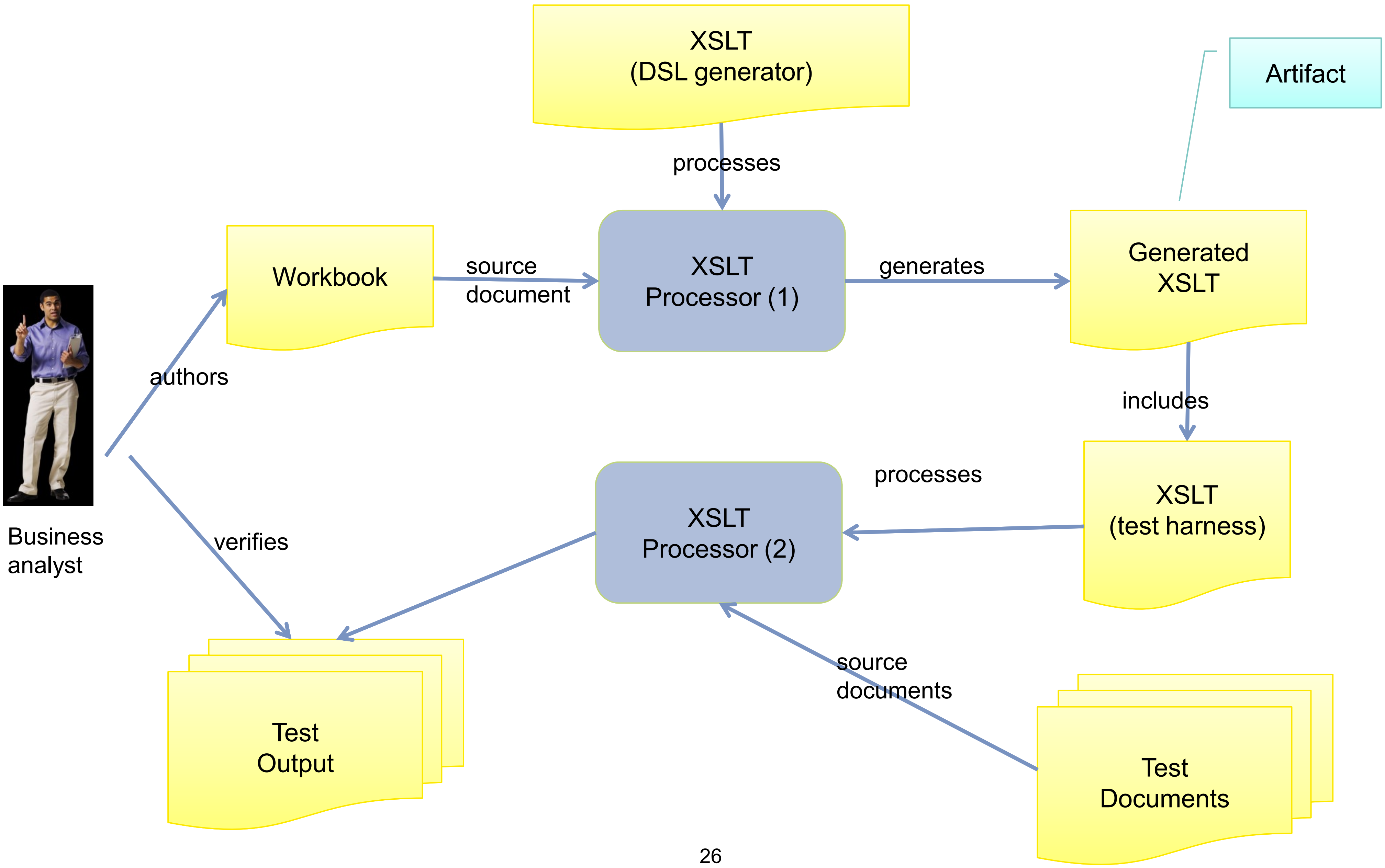


Spreadsheet DSL For Extracting Tabular Data



	A	B	C	D	E	F	G
1	namespace	fpml	http://www.fpml.org/FpML-5/recordkeeping				
2	namespace	fixml	http://www.fixprotocol.org/FIXML-4-4				
3							
4	default-prefix			fixml	fpml		
5							
6	header	number/type	name	***FixmlBond***	***FpmlBond***	#FWD-SystemA	#FWD-SystemB
7	for-each			/Bond/TrdCaptRpt	/trade[details/bond]		
8	for-each				.[@system = ('SystemA', 'SystemB')]		
9	variable	element(fpml:trade)	**trade**		current()		
10	variable	element(fixml:TrdCaptRpt)	**trade**	current()			
11	variable	xs:string	**ourParty**		**$trade**/header/onBehalfOf/@id		
12	variable	xs:string	**book**	**$trade**/TrdLeg/@BookId	**$trade**/acct/book[@id eq **$ourParty**]/@ref		
13							
14	column	1	**currency**	**$trade**/ccy	**$trade**/leg[1]/@Ccy		
15	column	2	**amount**	**$trade**/@lastQty	**$trade**/@notional * 100		
16	column	3	**system**	'SystemC'	/*/@system		
17	column	4	**trader_id**	**$trade**/@primaryTrader	#FWD(/*/@system)	**$trade**/@userid	/*/details/trader
18	column	5	**end_date**	**$trade**/instr/@maturity	**$trade**/payments[last()]/@date		
19	column	6	**trading_book**	**$book**	**$book**		
20							

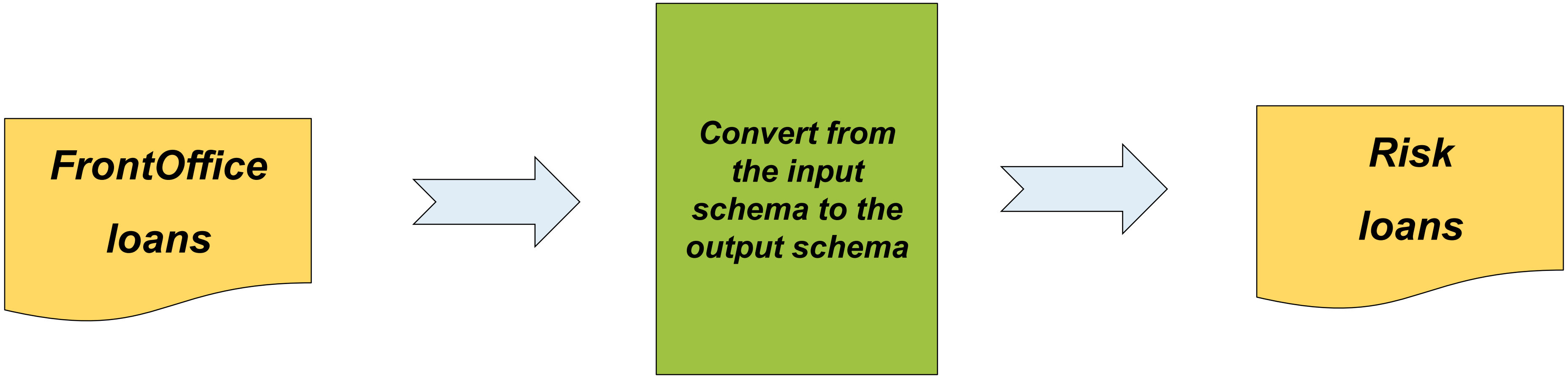
Steps in the Generation and Testing of the XSLT Artifact



Observed Results

- Good acceptance by the Business Analysts
  - BAs would even author Xpath functions in XSLT (e.g. sorts)
- Immediate testing results were a big benefit
- Some additional tools for analyzing data were also created (cardinality)
- Results are very structured with a *Rosetta Stone* type of equivalence

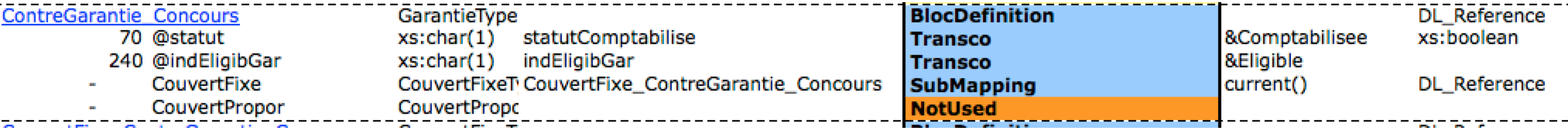
Schema-to-Schema Translation



- Globally very simple process (although some other flows not shown)
- The FrontOffice and Risk schemas were very different
  - Both strongly defined in XML Schema
  - Designed by different teams
  - Each had its own subject matter experts
  - Needed to find agreement between the two teams of subject matter experts

XSLT Templates for Schema-Aware Processing

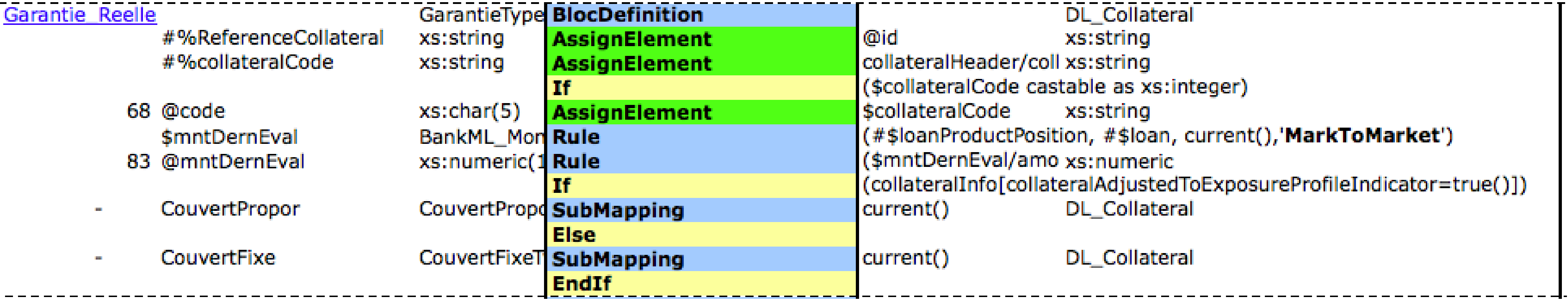
```
<!-- ================================= -->
<!-- ContreGarantie_Concours: (150)    -->
<!-- ================================= -->
<xsl:template match = "element(*,defiml:DL_Reference)"
              as    = "element(*, fsc2:GarantieType)"
              mode  = "ContreGarantie_Concours" >

    <xsl:param name="elementName" as="xs:string"                     required="yes"               />
    <xsl:param name="facility"    as="element(*,defiml:DL_Facility)*" required="yes" tunnel="yes" />
    <xsl:param name="loan"        as="element(*,defiml:DL_Loan)*"     required="yes" tunnel="yes" />

    <xsl:element name="{$elementName}" type="fsc2:GarantieType" >
        <xsl:attribute name="statut"       select="transco:statutComptabilise('Comptabilisee')" />
        <xsl:attribute name="indEligibGar" select="transco:indEligibGar('Eligible')" />

        <xsl:apply-templates select="current()" mode="CouvertFixe_ContreGarantie_Concours" >
            <xsl:with-param name="elementName" select="'CouvertFixe'" as="xs:string"/>
        </xsl:apply-templates>
```



ContreGarantie_Concours		GarantieType		**BlocDefinition**		DL_Reference
70	@statut	xs:char(1)	statutComptabilise	**Transco**	&Comptabilisee	xs:boolean
240	@indEligibGar	xs:char(1)	indEligibGar	**Transco**	&Eligible	
-	CouvertFixe	CouvertFixeT	CouvertFixe_ContreGarantie_Concours	**SubMapping**	current()	DL_Reference
-	CouvertPropor	CouvertPropc		**NotUsed**		

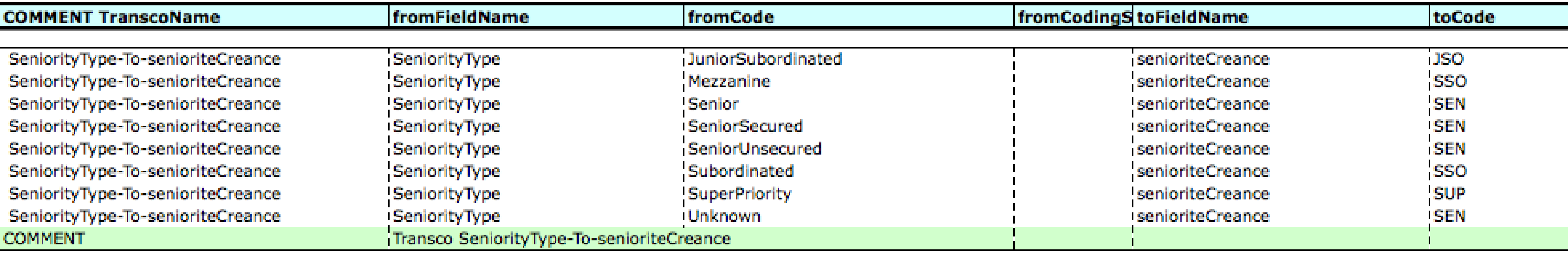
XSLT Templates for Schema-Aware Processing (2)

```
<!-- ================================= -->
<!-- Garantie_Reelle: (368)            -->
<!-- ================================= -->
<xsl:template match ="element(*,defiml:DL_Collateral)"
              as    = "element(*, fsc2:GarantieType)"
              mode  = "Garantie_Reelle" >
    <xsl:param name="elementName" as="xs:string"                required="yes"              />
    <xsl:param name="loan"        as="element(*,defiml:DL_Loan)*" required="yes" tunnel="yes" />
    <xsl:param name="loanProductPosition"                         required="yes" tunnel="yes
                                  as="element(*,defiml:DL_LoanProductPosition)*" "            />


    <xsl:variable name="collateralCode" as="xs:string"
                  select="collateralHeader/collateralGroupTypeCode[codingScheme='FIN_RSK']/code" />
    <xsl:variable name="ReferenceCollateral" as="xs:string" select="@id" />
    <xsl:attribute name="code" select="$collateralCode" />
    <xsl:variable name="mntDernEval" as="element(*,defiml:BankML_Money)"
                  select= "brkfct:getCollateralValuationAmount($loanProductPosition,
                                                               $loan, current(),'MarkToMarket'))" />
```



Garantie_Reelle		GarantieType	**BlocDefinition**		DL_Collateral
	#%ReferenceCollateral	xs:string	**AssignElement**	@id	xs:string
	#%collateralCode	xs:string	**AssignElement**	collateralHeader/coll	xs:string
			If	($collateralCode castable as xs:integer)	
68	@code	xs:char(5)	**AssignElement**	$collateralCode	xs:string
	$mntDernEval	BankML_Mon	**Rule**	(#$loanProductPosition, #$loan, current(),'**MarkToMarket**')	
83	@mntDernEval	xs:numeric(1	**Rule**	($mntDernEval/amo	xs:numeric
			If	(collateralInfo[collateralAdjustedToExposureProfileIndicator=true()])	
-	CouvertPropor	CouvertPropo	**SubMapping**	current()	DL_Collateral
			Else		
-	CouvertFixe	CouvertFixeT	**SubMapping**	current()	DL_Collateral
			EndIf		

Transcodification (Code List Translations)

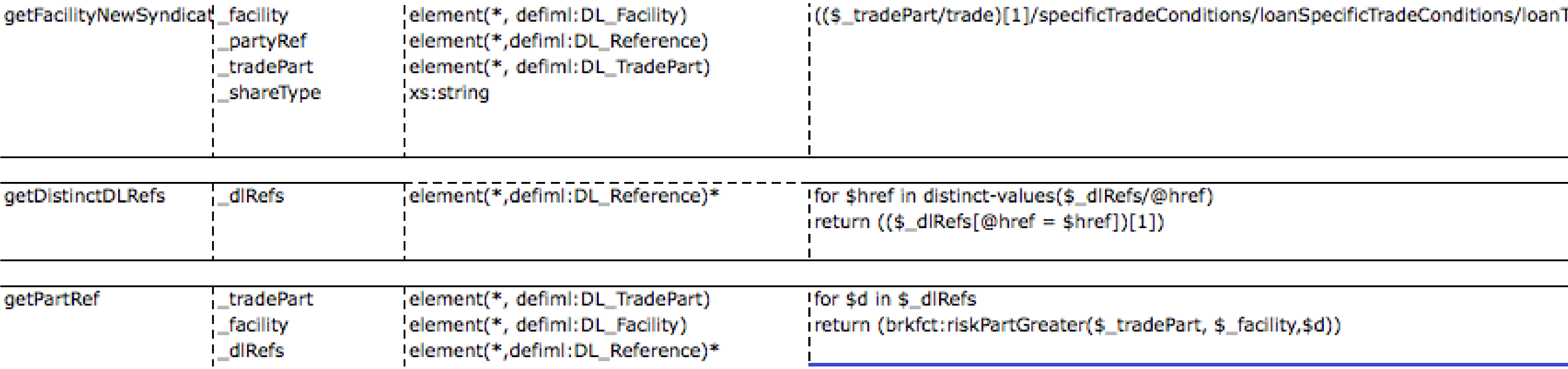


COMMENT TranscoName	fromFieldName	fromCode	fromCodingS	toFieldName	toCode
SeniorityType-To-senioriteCreance	SeniorityType	JuniorSubordinated		senioriteCreance	JSO
SeniorityType-To-senioriteCreance	SeniorityType	Mezzanine		senioriteCreance	SSO
SeniorityType-To-senioriteCreance	SeniorityType	Senior		senioriteCreance	SEN
SeniorityType-To-senioriteCreance	SeniorityType	SeniorSecured		senioriteCreance	SEN
SeniorityType-To-senioriteCreance	SeniorityType	SeniorUnsecured		senioriteCreance	SEN
SeniorityType-To-senioriteCreance	SeniorityType	Subordinated		senioriteCreance	SSO
SeniorityType-To-senioriteCreance	SeniorityType	SuperPriority		senioriteCreance	SUP
SeniorityType-To-senioriteCreance	SeniorityType	Unknown		senioriteCreance	SEN
COMMENT	Transco SeniorityType-To-senioriteCreance				

- BAs are in charge of the translations
- Could also pull these from an external system if available

```
<xsl:function name="transco:SeniorityType-To-senioriteCreance" as="xs:string">
  <xsl:param name="_simple" as="defiml:DL_SeniorityTypeScheme"/>
  <xsl:sequence select="transcoJ:transco('SeniorityType-To-senioriteCreance', $_simple))"/>
</xsl:function>
```

Rules (i.e. Xpath Functions)



getFacilityNewSyndicat	_facility
	_partyRef
	_tradePart
	_shareType
		element(*, defiml:DL_Facility)
		element(*,defiml:DL_Reference)
		element(*, defiml:DL_TradePart)
		xs:string
			(($_tradePart/trade)[1]/specificTradeConditions/loanSpecificTradeConditions/loanT

getDistinctDLRefs	_dlRefs	element(*,defiml:DL_Reference)*	for $href in distinct-values($_dlRefs/@href)
			return (($_dlRefs[@href = $href])[1])

getPartRef	_tradePart
	_facility
	_dlRefs
		element(*, defiml:DL_TradePart)
		element(*, defiml:DL_Facility)
		element(*,defiml:DL_Reference)*
			for $d in $_dlRefs
			return (brkfct:riskPartGreater($_tradePart, $_facility,$d))

- BAs could write these rules in the spreadsheet
- This could not handle everything (ex: sorting) but was largely used

```
<xsl:function name="brkfct:getDistinctDLRefs" as="element(*,defiml:DefiML_Reference)*" >
       <xsl:param name="_dlRefs" as="element(*,defiml:DefiML_Reference)*" />
       <xsl:sequence
            select="
               for $href in distinct-values($_dlRefs/@href)
               return (($_dlRefs[@href = $href])[1])
            "/>
   </xsl:function>
```

Observed Results

- Business Analysts were able to start with the model very early in the project
  - Detailed Specifications, Rules and Transcodifications authored originally in the DSL
- Immediate testing results were a big benefit (again)
- Subject matter experts (SMEs) used the DSL in meetings (often printed)
- SMEs also used an additional column in the DSL to indicate if they had validated each individual rule (fine-grained validation)
- The approach was quickly adopted for a number of other flows including a reverse flow

Some Tentative Conclusions

- The DSL Representation is extremely useful in the short and in the long run
- I’ve found Business Analysts to be mostly positive on the approach
  - Some BAs do not want to have to work on a « technical level »
  - In these cases, can transcribe any BA work into the DSL and then agree upon using the DSL as the common support for ongoing work
- The development time on the DSL is not that important (a few days of work)

- Designing a DSL does require creativity and some vision
- The technical implementors need to be enthusiastic about the approach
  - Their enthusiasm will win over recalcitrant SMEs and BAs

What Can’t Be a DSL in a Spreadsheet?

- I haven’t identified anything intrinsically too structured to be represented as a DSL in a Spreadsheet

- I do have a conjecture:
  - “Any functional process can be represented as a DSL in a Spreadsheet”***

  - *** “provided that the implementor is clever enough”

Caveats

- Spreadsheet documents can be difficult for source control systems (ex: git)
  - Can’t merge two divergent branches very easily
  - Also can’t display differences between successive versions in a branch

Thanks for Listening

Questions?