APPLYING MODEL-DRIVEN ARCHITECTURE AND SPARK ADA

A SPARK Ada Model Compiler for xtUML

Erik Wedin, Senior Specialist – Software Systems Architecture
Saab Bofors Dynamics AB, Sweden
erik.wedin@saabgroup.com

June 16th 2010

15th International Conference on Reliable Software Technologies – Ada-Europe 2010

CONTENTS

SAAB

EXECUTABLE UML AND MDA

THE SPARK REQUIREMENT

SPARK SOFTWARE ARCHITECTURE

SPARK ADA xtUML MODEL COMPILER

xtUML MODEL AND GENERATED CODE EXAMPLE

CURRENT STATUS

CONCLUSIONS AND OBSERVATIONS

WE ALWAYS STRIVE TO BE AT THE FOREFRONT OF CHANGE

1941	1948	1979	1990	1993	2002	2005	2006	2008
First B17 delivered	Tunnan – first flight		First laser simulator BT46	First Gripen delivered	First contract for NLAW	Contract for Neuron	Saab 2000 ERIEYE™ AEW&C	Gripen Demo – first flight

SAAB

- **1937** Saab is founded
- **1990** Saab Automobile independent company
- **2000** Saab acquires Celsius
- **2005** Saab acquires Grintek
- **2006** Saab acquires EMW

1646	1894	1948	1998
Bofors Järnbruk is founded	Alfred Nobel acquire Bofors	First order for Carl Gustaf	StriC in operation

1950-	1970-	1980-	1990-
Development of fighter radar	Development of GIRAFFE	Development of ARTHUR	Sea Giraffe AMB is launched

COMPANY UNCLASSIFIED

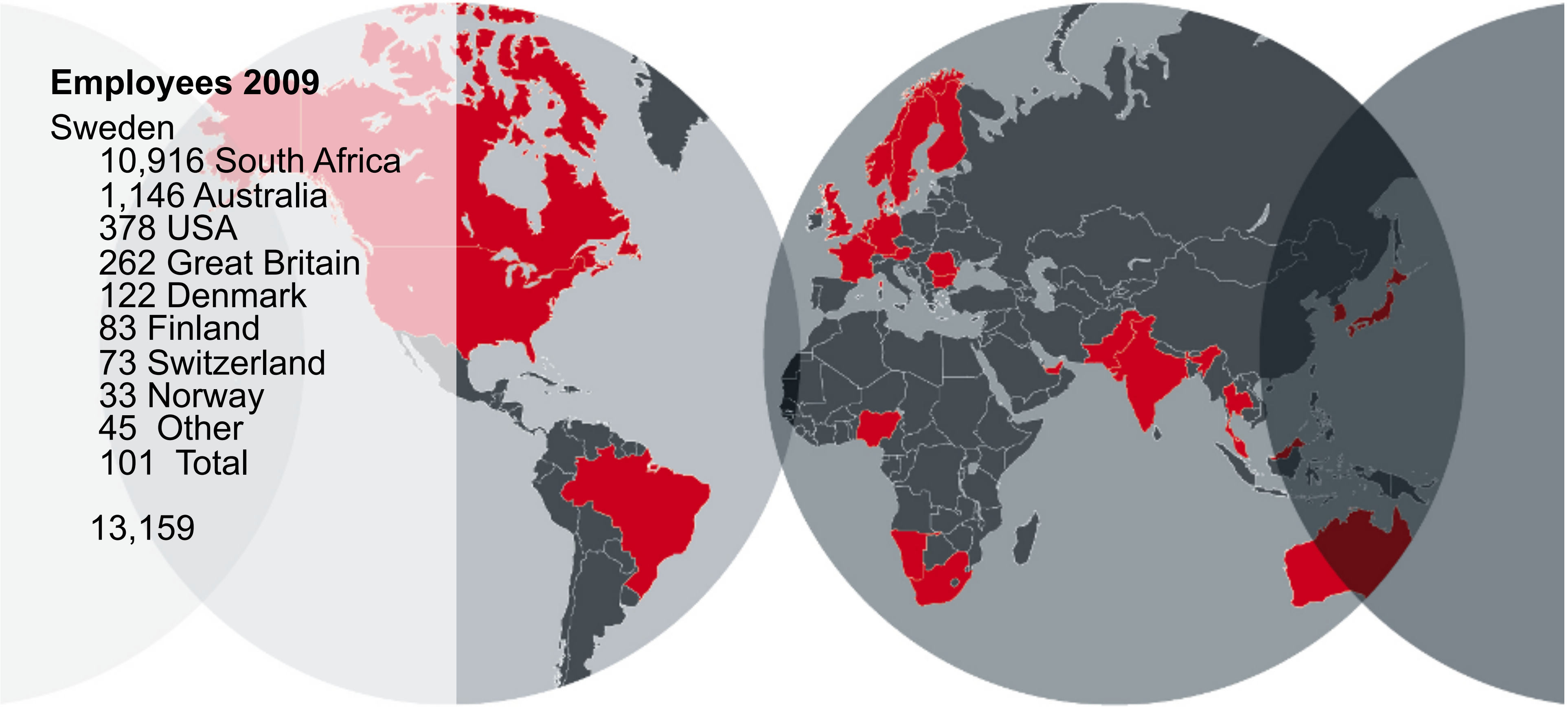
SAAB

SAAB WORLDWIDE

Employees 2009

Sweden
10,916 South Africa
1,146 Australia
378 USA
262 Great Britain
122 Denmark
83 Finland
73 Switzerland
33 Norway
45 Other
101 Total

13,159



DYNAMICS

Business portfolio:

- Support weapons
- Missiles
- Torpedoes and ROV (Remotely Operated Vehicle) and AUV (Autonomous Underwater Vehicle)
- Signature Management Systems
- Headed by Tomas Samuelsson

MSEK	Jan-Dec 2009	Jan-Dec 2008	Jan-Dec 2007
Order intake	3,133	3,743	3,870
Order backlog	6,980	8,453	8,882
Sales	4,580	4,281	3,812
EBITDA	466	497	494
EBITDA margin, %	*10.2*	*11.6*	*13.0*
Operating income	269	112	406
Operating margin, %	*5.9*	*2.6*	*10.7*
Adj. Oper. Margin, % (excl. non-recurring items)	*9.8*	*8.5*	*9.5*
Operating cash flow	369	830	-822
Number of employees	1,739	1,805	1,849
Split sales in Sweden and markets outside of Sweden	14/86	19/81	25/75

proforma financials

SAAB

COMPLETE MISSILES SOLUTIONS

- Develops advanced missile systems for the Swedish Defence Forces and other national defence forces
- Participates in international projects

EXECUTABLE UML AND MDA

- Executable UML is a profile (subset) of UML 2.0, including an abstract action language, adhering to the now standardised Action Semantics – defined by Stephen Mellor/Marc Balcer in 2002
- Enables development of Software and Hardware platform-independent specifications of the problem
- A standardised UML action language syntax is about to be defined – Executable UML is the basis for that via Stephen Mellor
- Supports the OMG Model-Driven Architecture (MDA) initiative
  - PIM – Platform-Independent Model- models the solution of a problem
  - PSM – Platform-Specific Model - models the details of the implementation
  - Separation of Subject Matters ➔ Abstraction & Reuse of models (not code)
  - A Model Compiler weaves the models together, guided by marks, and translates them into an implementation at *design-time* not at *specification-time*
- xtUML is Mentor Graphics' implementation of Executable UML
- Executable UML (xtUML) models ≡ Executable Specifications
  - can be executed and simulated (platform-independently) – *without generating code*
  - can be translated to one/several implementation(s) onto one/several specific software/ hardware platform(s) – *without changing the models*

MODEL COMPILERS

- Can be bought, tailored or developed from scratch
  - depending on architectural requirements, e.g. required implementation language and target platform
- Main components
  - **architecture metamodels** – expressed in xtUML, formalising architectural properties
  - **marks** – translation control used to direct the translation to use different translation rules to inject design decisions during the translation, expressed in a rule-specification language (RSL)
  - **archetypes** – translation rules and templates querying and transforming the information in the populated metamodels, expressed in RSL – Rule Specification Language
  - **mechanisms** – library components expressed in the target source code language, e.g. event handler, timer
- BridgePoint supports full open code translation which means that the user have full control over the translation/ code generation process → **<u>Key property</u>!**

THE SPARK REQUIREMENT (I)

- Joint programme with a partner company
  - development of embedded real-time software
  - overall software is safety-related so the main parts of the software are implemented in SPARK Ada
  - the partner company is a long-time user of SPARK
- Initial development approach
  - Saab delivered non-safety-related software components in source code
  - non-safety-related components temporally separated from the execution of the safety-related software
  - specifying/modelling functionality in xtUML
  - using Saab's own Ada Model Compiler – generating full Ada
    - slightly modified to generate SPARK-compliant interface layer to support overall SPARK analysis
  - software successfully integrated and deployed by the partner in a number of builds and used in live trials

THE SPARK REQUIREMENT (II)

- Changed safety requirement
  - temporal separation removed
  - execution of non-safety related functions concurrently with the safety-related part of the application
  - Saab’s code had to be implemented in SPARK Ada
- Revised development approach
  - MDA process retained
  - reuse existing xtUML models as-is
  - develop a new software architecture in SPARK Ada
  - formalise it into an xtUML model compiler based on SPARK
  - reuse the Ada Model Compiler design as far as possible
  - **The Problem** – Saab lacking sufficient in-depth knowledge of SPARK
  - **The Solution** – form a joint architectural design team of MDA/xtUML/Model Compiler experts from Saab and SPARK experts from the partner

SPARK SOFTWARE ARCHITECTURE REQUIREMENTS

- Model-driven
  - generate 100% complete SPARK code & annotations from xtUML models
- xtUML model compatibility
  - no changes in existing xtUML application models
- xtUML feature support
  - support all executable diagrams
- SPARK analysis support
  - generate full code and annotations to support
    - dataflow analysis
    - information flow analysis
    - proof of absence of run-time errors
- Annotation approach
  - SPARK used to show generated code is structurally sound
  - annotations relate to architectural elements rather than application functionality
- High-performance
  - the code should be fast & small
- Integrity
  - <10% remaining unsimplified Verification Conditions (VCs) from the Proof-of-Absence-of-RTE analysis
- Minimise requirement for SPARK knowledge and training
  - simple mechanical process to compute annotations
- xtUML action language support
  - try not to restrict the use of the action language

SPARK SOFTWARE ARCHITECTURE DESIGN APPROACH

- 13 technical workshops – 4 days each, ~4 engineers
  - Saab xtUML/Model compiler experts + partner SPARK experts
- Prototype xtUML application model covering most xtUML modelling constructs
  - base for the software architecture design
- Prototype model manually implemented in SPARK
  - including annotations
  - explore design options and how best to annotate
  - implementation patterns designed and redesigned for each xtUML model construct, integrated and tested together
  - iterative development
  - put through static SPARK analysis and dynamic tests
  - static properties, like integrity, and dynamic properties, like execution performance were proven and fed back into next iteration
- Result: SPARK Ada software architecture suitable for automatic translation from xtUML

SPARK SOFTWARE ARCHITECTURE DESIGN NOTES – ADA CODE

- Extensive use of subtypes
  - keep efficient underlying base type and to get good simplification of VCs
- Encountered conflicts between SPARK and the structuring of the design
  - parent-child hierarchies → visibility within parent-child
  - state refinement → hierarchies
- Constant look-up tables not always simplified by the SPARK Simplifier
  - even when they cannot lead to runtime errors.
- Some preconditions were added in the annotations
  - propagate the encountered issue to an appropriate higher level. But, as yet, no post conditions have been required.
- xtUML implicit declaration of local variables
  - in the block scope where they are assigned, e.g. in the else-branch in an if-statement
  - implicit block structure of action language needed to be reflected in the SPARK code
- Forced initialisation of action language variables
  - unnecessary explicit local variable assignments whose only purpose is to declare a local variable had to be detected and removed by the model compiler
  - SPARK lead to an efficiency gain due to removal of unnecessary initialisations

SPARK SOFTWARE ARCHITECTURE DESIGN NOTES – ANNOTATIONS

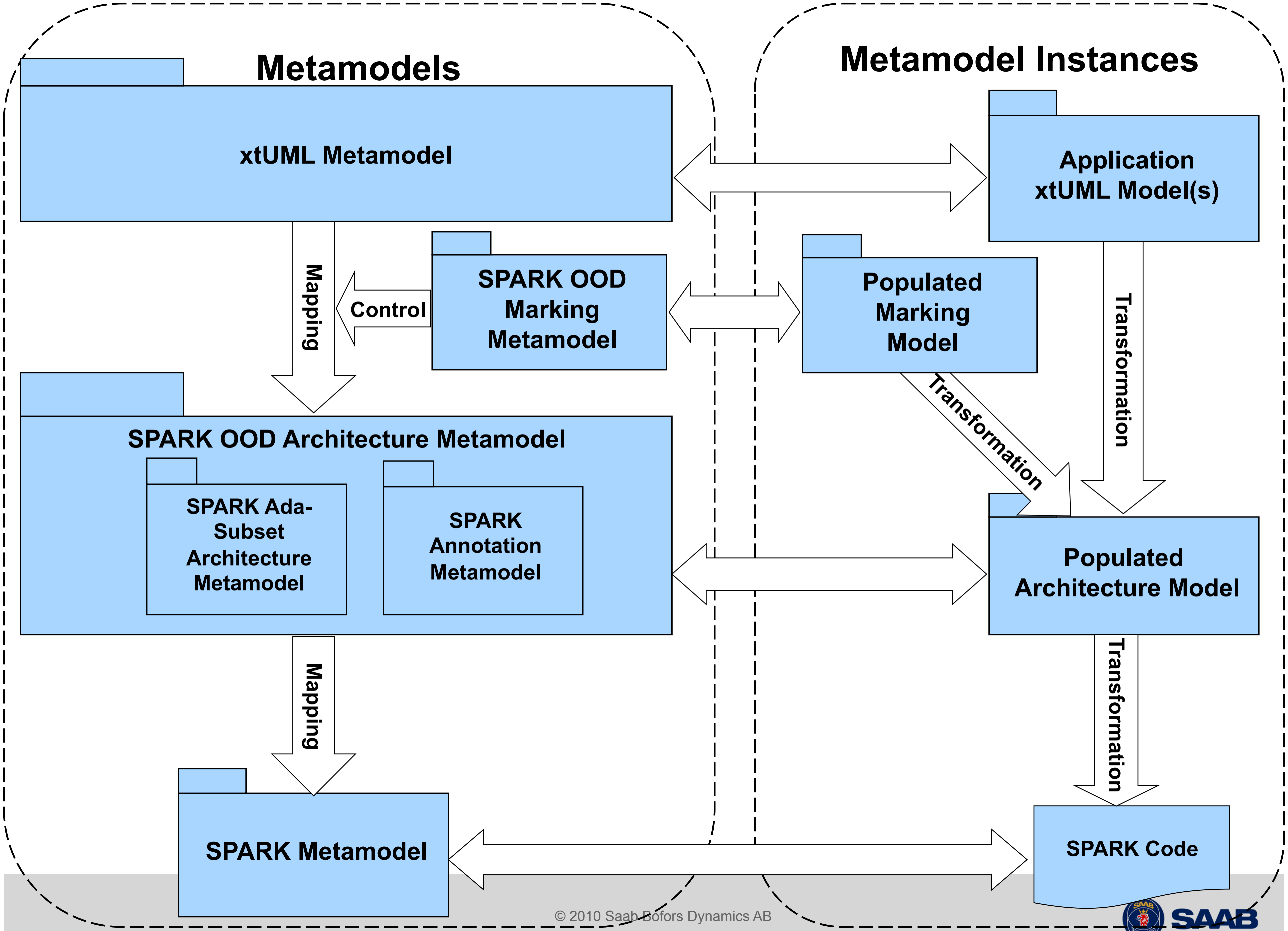
- Prototype annotations developed as in a manual development
- Prototype application designed to exploit xtUML constructs-of-interest
  - all hand-coded
  - used to assess efficiency and the proof of absence of run-time errors
  - remaining unsimplified Verification Conditions ➔ ~5% which was surprisingly good!
- Annotations were “computed” by keeping track of variable usage and was to be produced by the model compiler

SPARK ADA xtUML MODEL COMPILER DESIGN

- Prototyped software architecture formalised into a model compiler
- Reuse of architecture metamodels covering basic xtUML features and marking archetypes from the pre-cursing full Ada Model Compiler
- New metamodels for SPARK architecture specific features
- SPARK Ada additional semantics formalised into metamodels, e.g.
  - global state
  - data and information flow – dependency relationships between package state and subprogram parameters
- Ada semantics formalised into metamodels, e.g.
  - package hierarchy
  - with-dependencies
  - package-subprogram relationship
  - subprogram invocations
- Structural design decisions formalised into metamodels, e.g.
  - one Ada package per class with operations in child-packages

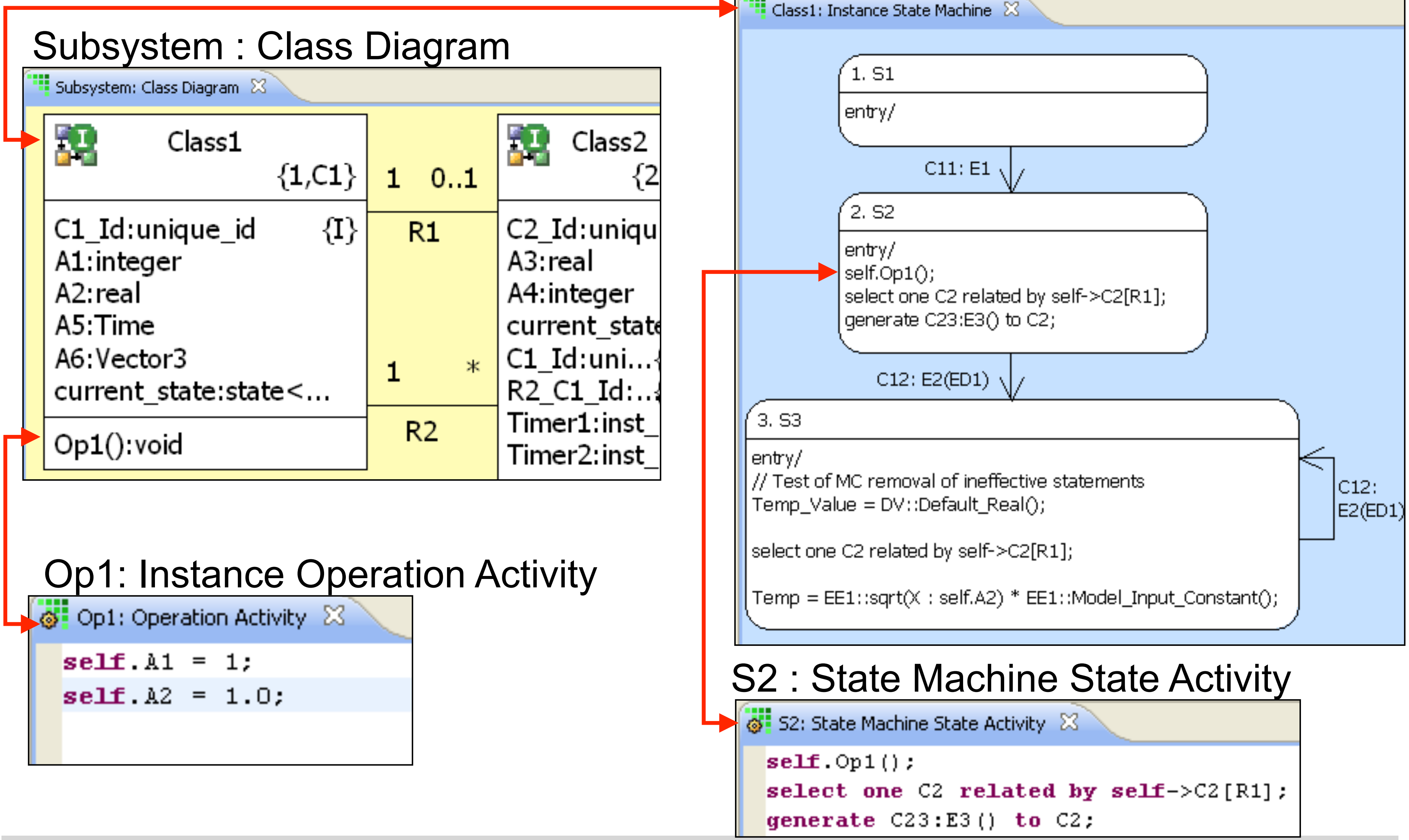
SPARK ADA xtUML MODEL COMPILER DESIGN

- Translation rules
  - translating xtUML application models into populated metamodels
  - then into source code were
  - formalised as new archetypes expressed in BridgePoint RSL (Rule-Specification Language)
- Marks
  - new marks to control the annotations of the generated subprograms that interfaces to realised components not modelled in xtUML (=code)
  - the model compiler does otherwise not have any information about that external software
- Mechanisms
  - none reused from the Ada Model Compiler
  - only a couple of new mechanisms were implemented
  - the rest are generated due to the lack of generics in SPARK


Metamodels
Metamodel Instances
xtUML Metamodel
Application xtUML Model(s)
Mapping
Control
SPARK OOD Marking Metamodel
Populated Marking Model
Transformation
Transformation
SPARK OOD Architecture Metamodel
SPARK Ada-Subset Architecture Metamodel
SPARK Annotation Metamodel
Populated Architecture Model
Mapping
Transformation
SPARK Metamodel
SPARK Code

SAMPLE xtUML MODEL

Class1 : Instance State Machine



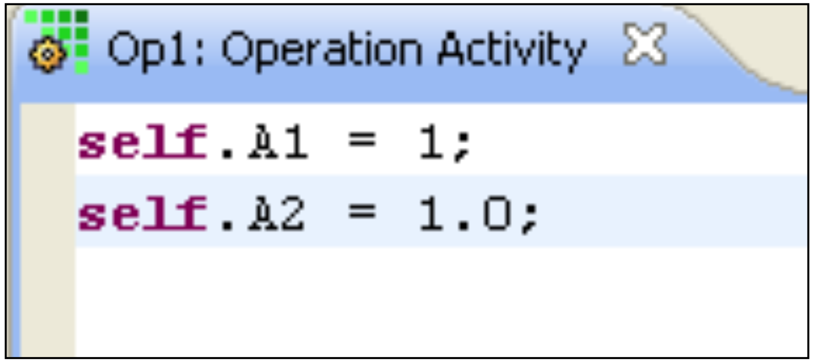
OPERATION OP1 – GENERATED CODE & ANNOTATIONS

```
with Standard_Types,
     D1_Domain,
     Process_1.D1.Class1,
     Process_1.D1.Class1.IAttr;
--# inherit Process_1.D1.Class1.IAttr,
--#         Standard_Types,
--#         D1_Domain,
--#         Process_1.D1.Class1;

package Process_1.D1.Class1.IOp.Op1
is
  procedure Invoke
    (Self : in Class1.Instance_Id);
  --# global in out IAttr.State;
  --# derives IAttr.State from *,
  --#                          Self;
private

end Process_1.D1.Class1.IOp.Op1;
```

```
package body
Process_1.D1.Class1.IOp.Op1
is
  procedure Invoke
    (Self : in Class1.Instance_Id)
  is
  begin
    IAttr.Set_A1(Self.Index, 1);
    IAttr.Set_A2(Self.Index, 1.0);
  end Invoke;
end Process_1.D1.Class1.IOp.Op1;
```



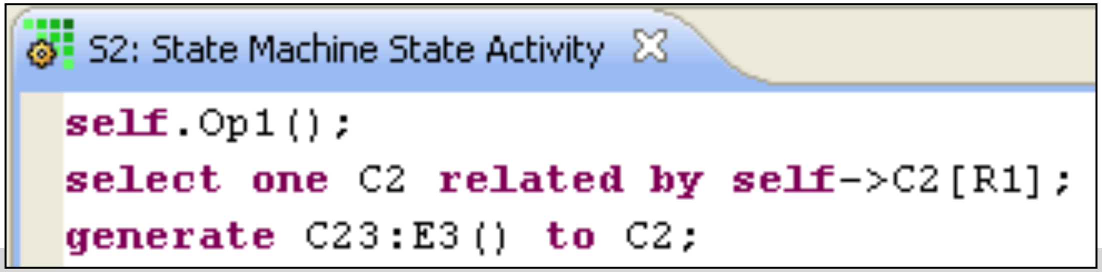
STATE S2 – GENERATED CODE & ANNOTATIONS

```
with Process_1.D1.Class1,
...
--# inherit Process_1.D1.Class1.IAttr,
...
package Process_1.D1.Class1.ISA
is
  procedure S2_Action
    (Self : in Class1.Instance_Id);
  --# global in D1_Domain.R1.State;
  --#        in out IAttr.State;
  --#        in out IEH.State;
  --# derives IAttr.State from *,
  --#           Self &
  --#          IEH.State from *,
  --#           Self,
  --#           D1_Domain.R1.State;
...
private

end Process_1.D1.Class1.ISA;
```

```
...
package body Process_1.D1.Class1.ISA
is
  procedure S2_Action
    (Self : in Class1.Instance_Id)
  is
    C2_Id_1 : Class2.Instance_Id;
  begin
    IOp.Op1.Invoke(Self);
    C2_Id_1 :=
      D1_Domain.R1.Class1_To_Class2.
        Select_One(Self.Index);
    IEH.Generate_No_Data
      (Event    => Process_1.D1_Class2_E3,
       Receiver =>
         Class2.I.Generalize(C2_Id_1),
       Sender   => I.Generalize(Self));
  end S2_Action;
...
end Process_1.D1.Class1.ISA;
```

ISA = Instance state action package
IOp = Instance operation package
IEH = Event handler package
IAttr = Attribute data package
I = Instance data package
R1 = Relationship R1 package



CURRENT STATUS

- Computationally-demanding and state logic models have been generated, analysed and integrated
- The SPARK Ada model compiler has been used to generate code for the real system, as planned. The code has passed both static analysis and dynamic tests
- The model compiler is a mature SPARK software architecture

CONCLUSIONS AND OBSERVATIONS (I)

- Data and information flow analysis
  - relatively easy to achieve; surprisingly useful
  - information flow analysis – substantially harder but achieved
- SPARK warnings and errors fall into clear patterns
  - easy to relate to the model
  - anticipated real issue here – but no.
- Reflection at initial application of the model compiler
  - xtUML modellers have the same types of issues as found in conventional SPARK development when specifying software based on a system/algorithm specification
  - an early system-software dialogue was prompted – which is a key benefit of SPARK.
- Was SPARK just some additional bureaucracy?
  - no, it added real value for zero effort – big win for both the system and software development – model update ⇔ SPARK code generated
  - properties from the SPARK code could be easily fed back to the algorithm system developers
- Generate-analyse
  - single combined step in the modelling process.
- RavenSPARK profile
  - used because of the need to access Ada real-time
  - analysis is not performed across partitions – each task is independent (so far)
  - an xtUML component is mapped to a task. Several components can be executed by the same task.
- Execution performance
  - the order found as semi-restricted Ada – at immature (but useful) state
  - the team wants to explore possible improvements

CONCLUSIONS AND OBSERVATIONS (I)

- xtUML modelling
  - encourages relatively more classes and relationships while using relatively small actions
  - the design was driven to be efficient for such models
  - made a good match with SPARK
- xtUML typing
  - currently too weak
  - Ada style typing would be beneficial
- Tension between flexibility for the modellers and the desire for tighter semantics
- Translating existing xtUML application models
  - minimum effort
  - mainly coping with the implicit declaration of local variables in the action language ➔ gives “ineffective statement” errors in the SPARK analysis
  - some of the assignments had to be substituted to a special value that the model compiler could identify
- Translation and code generation time
  - an issue, mainly related to the information flow traversals
  - probably largely resolved when migrating model compiler action translation to the latest metamodel-based technique
- Effort to date
  - ~1 man-year over a period of 2-3 years [4 engineers]
  - performed part-time in parallel with project application
  - development has been slotted into the main programme

SAABGROUP.COM