SysML Modelling Language explained

- **Date**: 7th October 2010
- **Author:** Guillaume FINANCE, Objet Direct Analyst & Consultant

UML, the standard modelling language used in the field of software engineering, has been tailored to define a modelling language for systems: **SysML** or **Systems Modeling Language**. This article is intended to provide a non-exhaustive presentation of SysML including some background about Systems Engineering and SysML, and a review of each SysML diagram and modelling techniques.

Systems Engineering

Systems Engineering (**SE**), is an approach and discipline to deal with complex systems realised through software and hardware solutions.

Systems Engineering applies to the following areas and industries: embedded systems (e.g. audio and video encoding/decoding, set top box), automotive, rail, aerospace, military, telecom, healthcare, energy, etc.

Systems Engineering (SE) relies on **modelling and simulation methods** to validate requirements or to evaluate the system. Therefore modelling is common practice in SE to deliver functional specifications, data flow description, or system structure definitions through the use of modelling techniques such as:

- Data Flow Diagram (DFD) to define the data going through a system, and any processing required
- Functional Flow Block Diagram (FFBD), similar to the UML activity diagram

Specifications produced with SE often result from a **document-based approach**, producing a large amount of documents with various types of diagrams or notations, sometimes used in an inconsistent manner.

The suggested alternative is the use of a **model-based approach**, usually referred as the “Model-Based Systems Engineering” or **MBSE**, to deliver a consistent set of system views, stored and managed in a *repository* such as Sparx Enterprise Architect.

The model-based approach leads to a structured set of models, representing and specifying the system at various levels of granularity such as the operational, functional, and technical aspects. Modelling the system helps in managing its complexity, since each model and diagram provides an abstracted view and definition of the system (or part of it).

SysML background

UML is the standard modelling language used in the software community. Even though UML should be sufficient to address Systems Engineering needs through its wide range of notations, it is recommended to adapt it with extensions defined with "UML profiles". As regards, several projects have been carried to create UML profiles for specific SE domains, e.g. MARTE and System on a Chip.

The decision to define a general-purpose modelling language based on UML and dedicated to SE was initiated in 2001 by the **INCOSE** (International Council on Systems Engineering) that got in touch with the **OMG** (Object Management Group) to form the Systems Engineering Domains Special Interest Group (SE DSIG). The goal of such language is defined as follows: "*A standard modelling language for systems engineering to analyze, specify, design, and verify complex systems, is intended to enhance systems quality, improve the ability to exchange systems engineering information amongst tools, and help bridge the semantic gap between systems, software, and other engineering disciplines*".

The INCOSE and the OMG were joined by members of the industry (BAE, Motorola, Boeing…), modelling tools vendors (IBM, Sparx Systems…), universities and organizations to define the systems modelling language **SysML**.

- 07/2006 : SysML is officially adopted by the OMG
- 09/2007 : SysML v1.0
- 12/2008 : SysML v1.1
- 06/2010 : SysML v1.2 (current version)

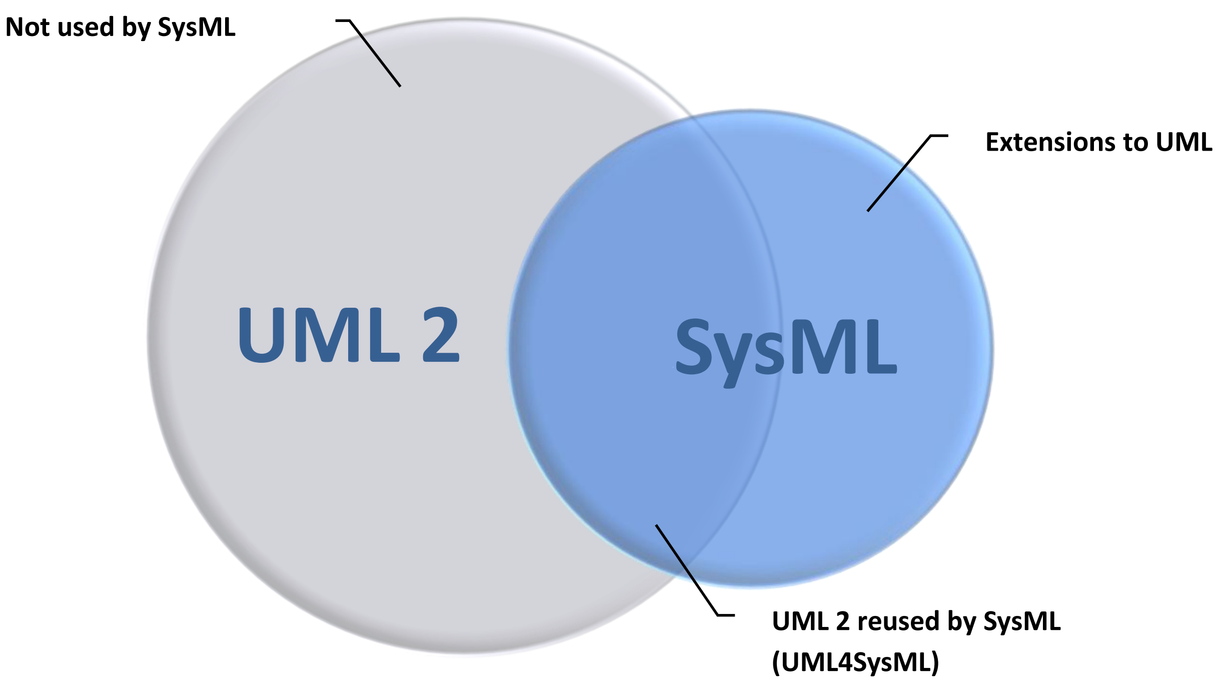
SysML overview

SysML is based on UML and involves **modelling blocks instead of modelling classes**, thus providing a vocabulary that's more suitable for Systems Engineering. A block encompasses software, hardware, data, processes, personnel, and facilities.

As specified on the following diagram, SysML reuses a subset of UML2 (UML4SysML), and defines its own extensions. Therefore SysML includes nine diagrams instead of the thirteen diagrams from UML2, making it a smaller language that is easier to learn and apply.

SysML can be easily understood by the software community, due to its relation with UML2, whilst it is accessible to other communities.

SysML makes it possible to generate specifications in a single language for heterogeneous teams, dealing with the realisation of the system hardware and software blocks. Knowledge is thereby captured through models stored in a single repository, enhancing communication throughout all teams. In the long term, blocks can be reused as their specifications and models enable suitability assessment for subsequent projects.



SysML defines the following diagrams:

- Structure diagrams
    - The **Block Definition Diagram** (BDD), replacing the UML2 class diagram
    - The **Internal Block Diagram** (IBD), replacing the UML2 composite structure diagram
    - The **Parametric Diagram**, a SysML extension to analyse critical system parameters
    - The **Package Diagram** remains unchanged
- Dynamic diagrams
    - The **activity diagram** has been slightly modified in SysML
    - The **sequence, state chart, and use case diagrams** remain unchanged
- The **requirements diagrams** is a SysML extension

This article continues with an overview of SysML, starting with the static diagrams, followed by the dynamic diagrams, and finishing modelling with new diagrams and modelling techniques introduced in SysML. This order (static modelling followed by dynamic modelling), doesn't relate to a process and methodology to apply on a project; this article is only intended to provide a presentation of SysML language, not the methodologies applicable to projects.

Structure modelling

BDD - Block Definition Diagram

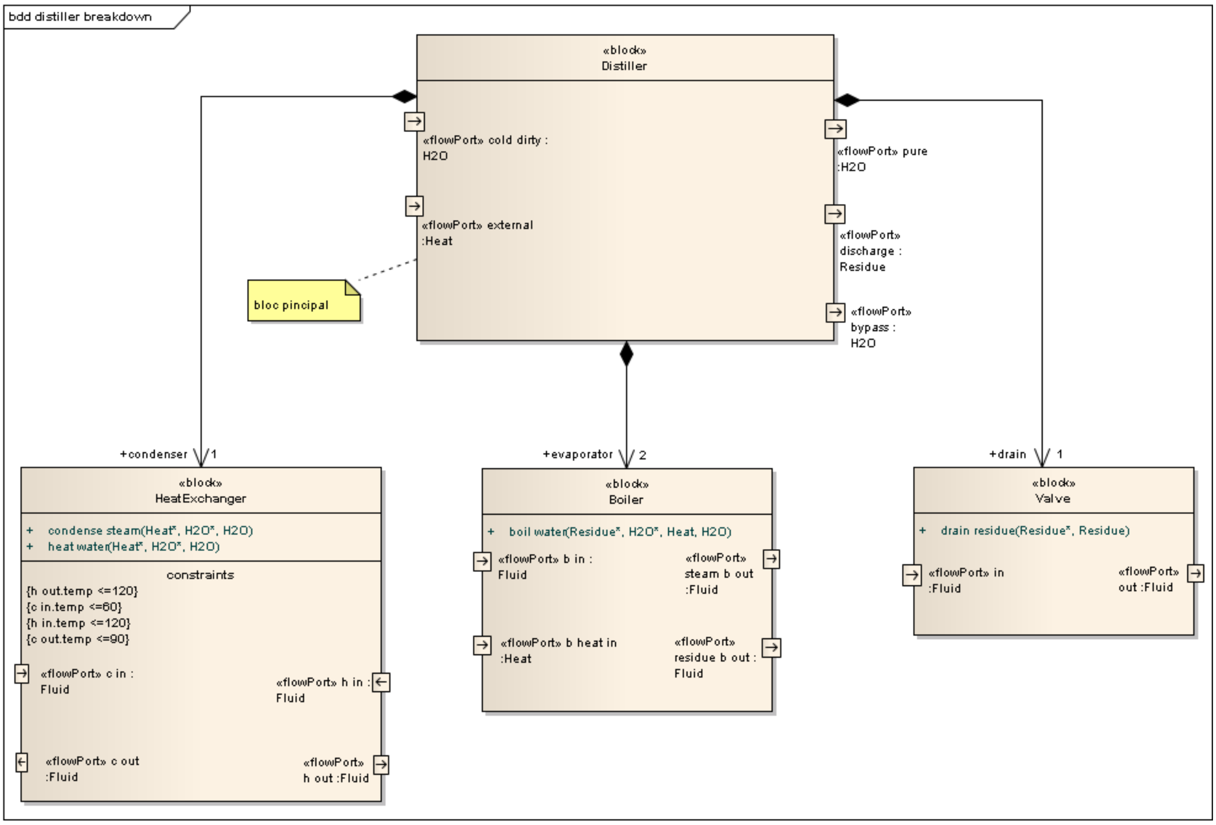
The Block Definition Diagram or BDD provides a black box representation of a system block i.e. **main block**, alongside the hierarchy of its composite blocks.

The BDD can include blocks of any type including software, hardware, etc.

The BDD can be compared with the first page of your furniture instructions, providing the contents inventory and the quantity of each part.

In comparison with UML2, the SysML BDD redefines the class diagram by replacing classes with blocks, and introducing flow ports (explained hereafter).

The following BDD corresponds to the OMG Distiller example.



BDD information summary:

- Blocks are shown as UML classes, stereotyped « block ».
- The main block defines the Distiller, itself composed by three types of blocks:
  - A heat exchanger that has a role of condenser
  - A boiler that has a role of evaporator (multiplicity indicates that 2 boilers will be instantiated)
  - A valve that has a role of drain
- These blocks physically belong to the main block ("has a" relationship), since the associations used on the diagram are compositions or "strong" aggregations, represented with a solid diamond. If a block was part of the main block but didn't physically belong to it, it would be called a *reference* and the association would be represented by an open diamond (simple aggregation).
- The flow port is a new definition from SysML. Flow ports represent what can go through a block (in and/or out), whereas it is data, matter, or energy. The above BDD specifies that the Distiller block takes as inputs cold water and external heat.

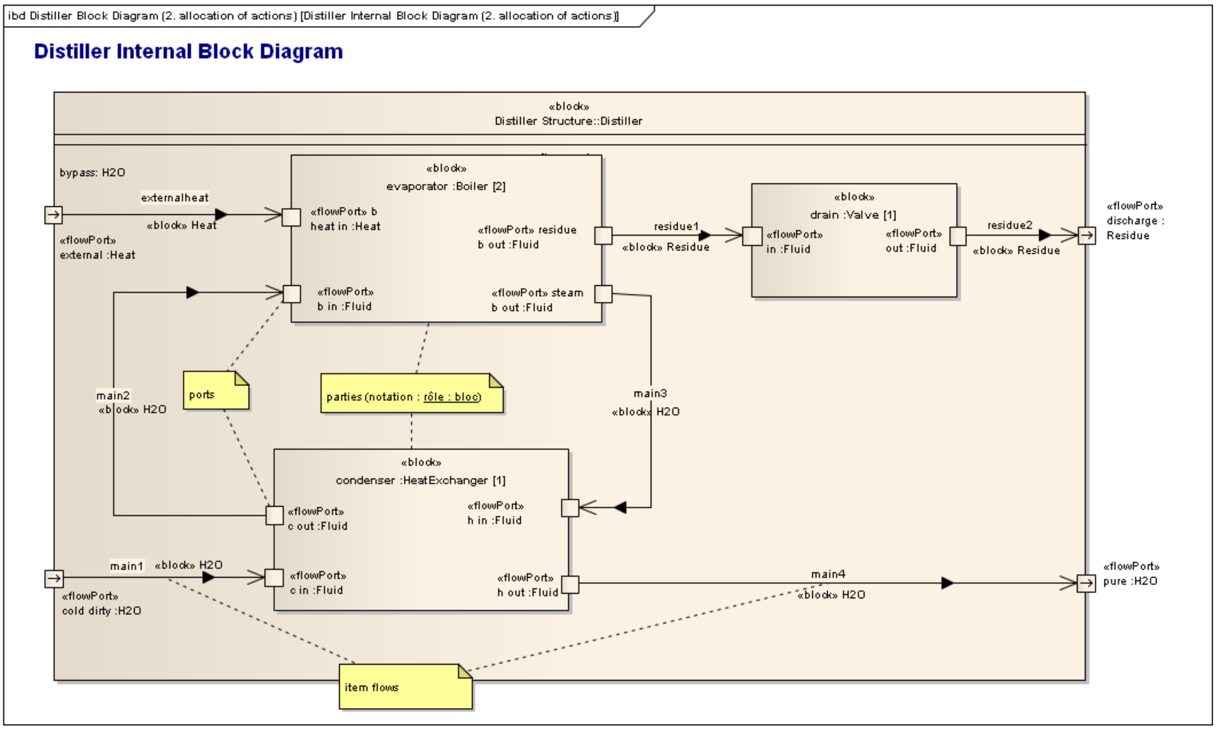
IBD - Internal Block Diagram

The Internal Block Diagram or IBD provides the white box or internal view of a system block, and is usually instantiated from the Block Definition Diagram (BDD) to represent the final assembly of all blocks within the main system block.

Composite blocks from the BDD are instantiated on the IBD as **parts**. These parts are assembled through **connectors**, linking them directly or via their **ports** (**standard ports** with exposed interfaces and/or **flow ports**).

In comparison with UML2, the SysML IBD redefines the composite structure diagram by supporting blocks and flow ports.

The following IBD corresponds to the OMG Distiller example, and is based on the previous BDD.



BDD information summary:

- The Distiller block of the BDD has been copied onto this diagram
- Blocks composing the Distiller specified in the BDD have been instantiated as **parts** on the IBD, and are entitled as follows: “role: name of the block [multiplicity]”.
- The role for a part must be consistent with the association ends on the BDD; hence the “drain” role specified on the BDD aggregation for the Valve block is consistent with the role name of the Valve part on the IBD.
- Multiplicity specified on BDD aggregations is also consistent with the IDB parts, where multiplicity is shown on the part within square brackets, e.g. « evaporator : Boiler [2] » (note: if there is no multiplicity on the part, it implies an undefined multiplicity or a default multiplicity of 1)

- The main block's ports (e.g. external: Heat, cold dirty: H2O) are all associated with ports of the internal parts via connectors. For instance the cold water coming in the Distiller feeds the internal Heat exchanger part.
- A flow port has a direction property that can be defined as an input, output, or input/output.
- Whereas standard and flow ports specify what can go through a block, **item flows** represent the things that flow between blocks and/or parts across their connectors. For instance the IBD specifies that the item "externalHeat" of type Heat (block), flows between the input flow port of the Distiller and the input flow port of the Boiler.

Value Types

Value Types are one of the new SysML extensions, and can be used as reusable types for properties or attributes in the model, for instance as types of Blocks' attributes. Similarly to UML where class attribute types can be of other classes in the model, SysML allows to define blocks property types with Value Types.

Value Types introduce a new concept via two optional properties: a **unit** and a **dimension** – e.g. the value type "°C" can be defined with unit = "Celsius degrees" and with no dimension. This would require "Celsius degrees" unit to be defined with dimension = "Temperature".

Package diagram

The package diagram allows defining the model structure as with UML.

Dynamic modelling

Modelling the behaviour of the system with SysML involves a selection of four UML2 diagrams: use case, sequence, activity, and state chart.

Among these diagrams, only the activity diagram has been slightly modified for SysML.

Use case diagrams

The same UML modelling techniques apply for SysML, where use case diagrams are intended to identify the actors and use cases from a usability perspective i.e. actors/system interactions.

Use cases define the system functional scope.

Sequence diagrams

A sequence diagram represents the items involved in a scenario or interaction, and the messages that are exchanged in a chronological order.

Items in a sequence diagram are represented by a **lifetime**. These lifetimes can be generic instances, or instances from blocks defined in the model. Instantiating blocks on sequence diagrams establish a link with the static system model, contributing towards the model consistency:

- A lifetime's Block can be accessed in the model repository (EA), to open its properties, BDD and any IBD
- Each message that is exchanged on the sequence diagram can be used to identify new block operations

All the sequence diagram definitions used in UML also apply to SysML: synchronous/asynchronous messages, operators (e.g. alt, loop; opt, par), references to other sequence diagrams, etc.

Activity diagram

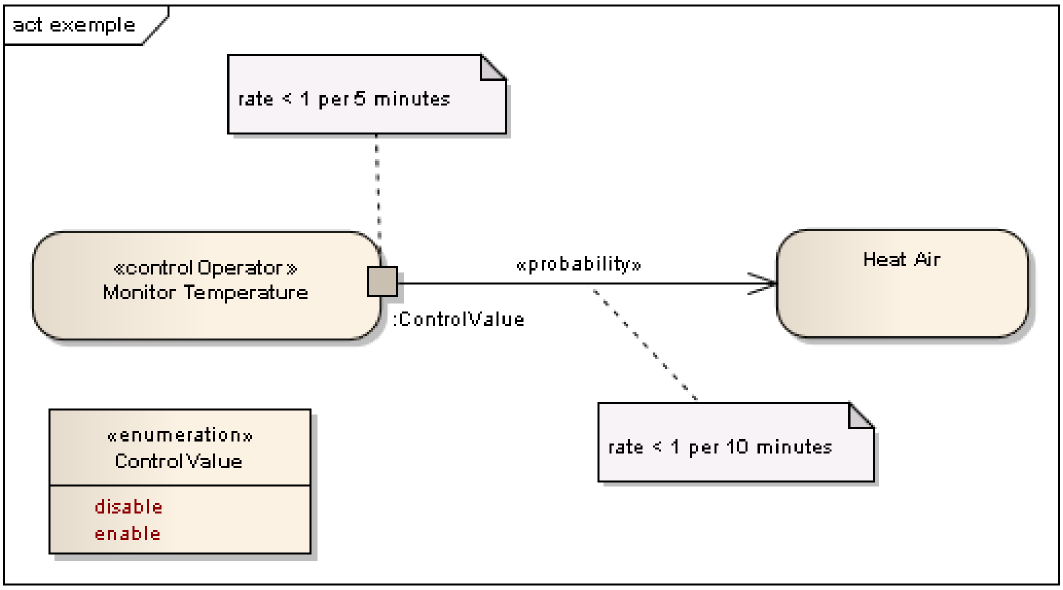
The activity diagram represents steps of a process, often making use of "input and output pins" that respectively correspond to the element type required as the input of an activity or action, and the element generated as an output.

If an action or activity corresponds to a block operation, it is possible to ensure that the types of the input and output of this activity are consistent with the block operation signature.

All the activity diagrams definitions used in UML also apply to SysML.

SysML has added a couple of extensions:

- With UML, control can only enable actions to start. SysML extends control to support disabling of actions that are already executing.
- Definition of the flow rate : continuous or discrete
- Definition of the rate and probability on the control or object flows



State chart diagrams

State chart diagrams are used as with UML2, i.e. they provide a way to define a Block lifecycle that all instances must comply with. A lifecycle defines all possible states for a block, and the events and conditions that define state transitions.

Only complex blocks, or important from a business perspective, should have a state chart diagram.

All the state chart diagram definitions used in UML also apply to SysML: events, guards/conditions, effects, transitions, composite states, regions, etc.

SysML extensions

Requirements

Both Systems Engineering and the software industry use requirements to formalize the stakeholders' needs, which will be realized as functionality and constraints, satisfied by the delivered application or system.

For the stakeholders, requirements are a mean to ensure that the solution (i.e. delivered system) is compliant with the list of requirements.

Requirements can be formalised and organized e.g. by separating functional from technical requirements. This can be achieved with an Excel spreadsheet, or using a dedicated tool such as a DOORS or EA. These tools have the advantage of letting users to fully manage and control requirements. A model based approach makes use of requirements through dependency associations with elements from the model such as use cases, blocks, or test cases, establishing the **model traceability**. For instance Enterprise Architect lets you define requirements in the model, or import them from other tools like DOORS, and create associations with elements of the model (e.g. a use case could have a realisation association with one or several requirements).

SysML specifications define the requirements modelling, by taking current features from the tools currently available on the market. Hence **SysML define a visual and graphical representation of textual requirements, specialised associations between themselves or with other elements of the model, and how they can be managed in a structured and hierarchical environment**.

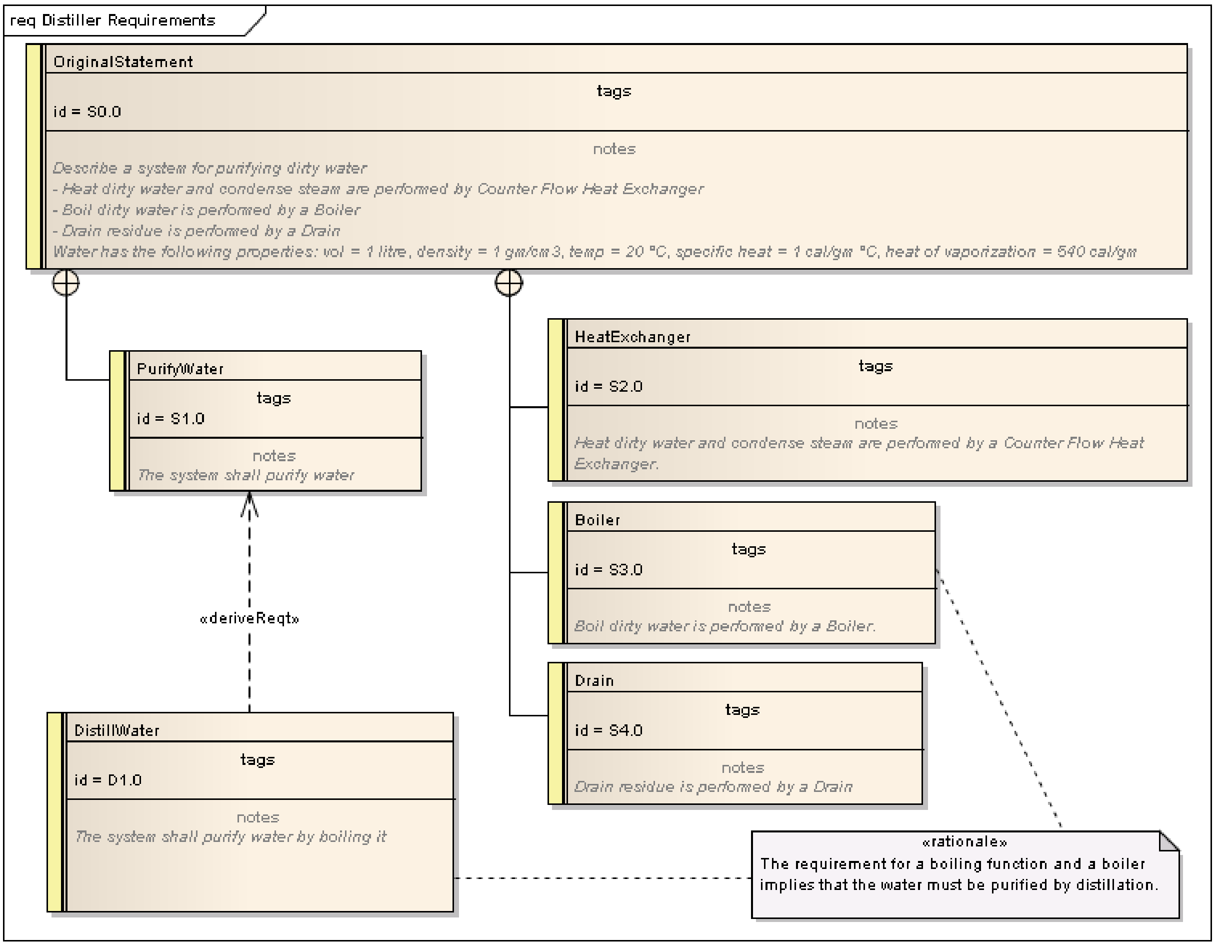
SysML defines new types of associations (stereotyped dependencies):

- *Derive* : one or several requirements derive from a requirement
- *Satisfy*: one or several model elements fulfil a requirement
- *Verify*: one or several model elements, e.g. a test case, verify that the system fulfils a requirement
- *Refine*: one or several model elements, e.g. a use case, further refine a requirement

SysML defines new types of comments, introducing stereotypes, enabling the association of explanations with associations or model elements:

- *Problem* : comment which description specifies the identified problem or need, following a deficiency, limitation, or failure of one or more model elements
- *Rationale*: comment which description provides the reason or justification on the decision related with the association or element

Illustration of SysML requirements modelled in EA for the Distiller system:



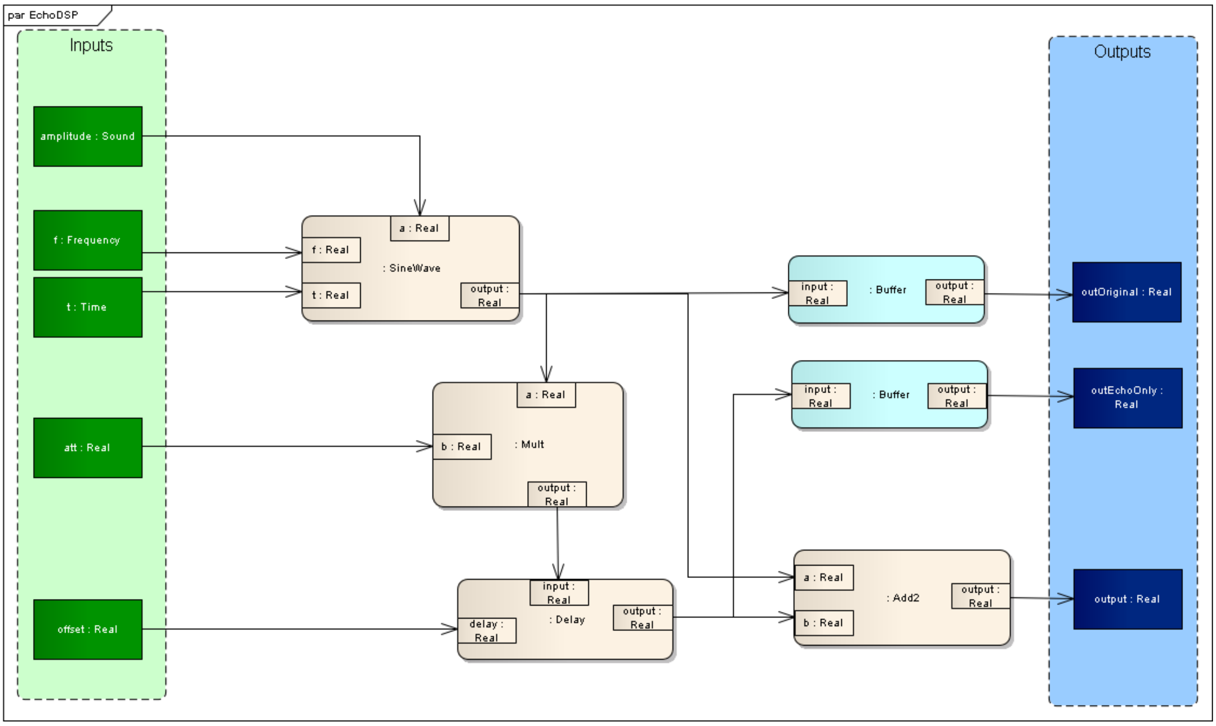
Parametric diagram

The parametric diagram is intended to support system analysis (performance, reliability, etc.) by defining **constraint blocks**. A constraint block expresses a mathematical equation and its parameters, some of which may correspond to system block properties.

To start with, similarly to the process of creating a BDD to define a block before creating the IBD, block constraints are defined in a class diagram. Once done, a parametric diagram can be created:

- Block constraints are instantiated as **constraint properties**, and inherit the parameters from the block constraint (note: there is no concept of input and output on these constraint parameters)
- System properties are added and may be associated with block properties
- Connectors are used to link all systems properties and constraint properties

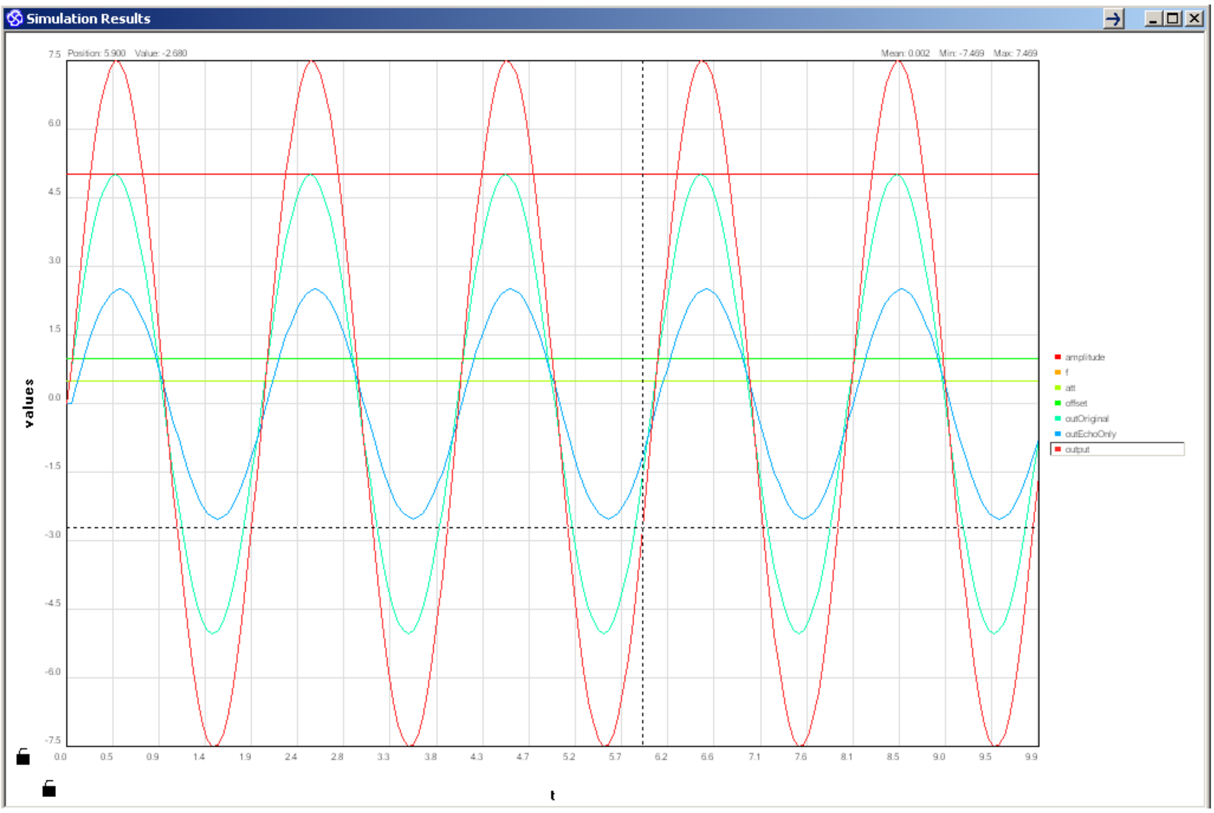
Parametric diagram copied from the MP3 player example given by Sparx Systems:



Parametric diagram information summary:

- The diagram contains six constraint properties of type SineWave, Mult, Delay, Add2, and Buffer (2)
- System parameters are split between inputs (green) and outputs (blue), and are associated with constraint property parameters (e.g. the input frequency f is linked with the f parameter of the constraint property SineWave)
- Some constraint properties are linked between each other (e.g. SineWare.output parameter is associated with Buffer.input, Mult.a and Add2.a)

Some tools provide an engine to use parametric diagrams for **simulation purposes**. Enterprise Architect lets you edit scripts in each constraint block (e.g. in VBScript or JavaScript), provide values for the input system parameters, and run the simulation on a graph, illustrated below, or to generate all output values in a spreadsheet.



Allocations

Allocation is a term taken from the systems engineers' vocabulary, and refers to a set of elements associated within a structured environment. Modelling a system requires attempts to allocate the various system elements. Once defined, allocations can be used and managed via a matrix to verify that all system parts are properly assembled.

Generating allocations enables consistency in the model, especially in the case of allocations between elements from the dynamic model and elements from the static model.

SysML tools

SysML language is supported by numerous commercial and open source tools.

Sparx Systems Enterprise Architect requires the SysML plug-in or the Ultimate version.