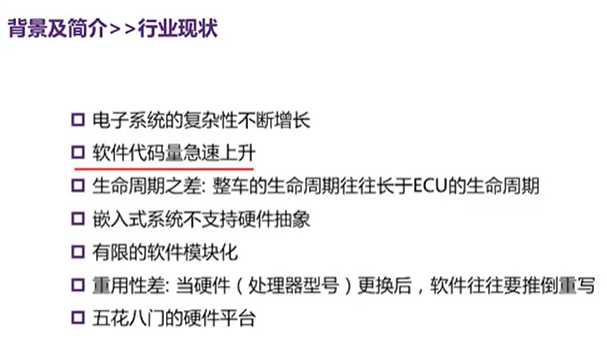
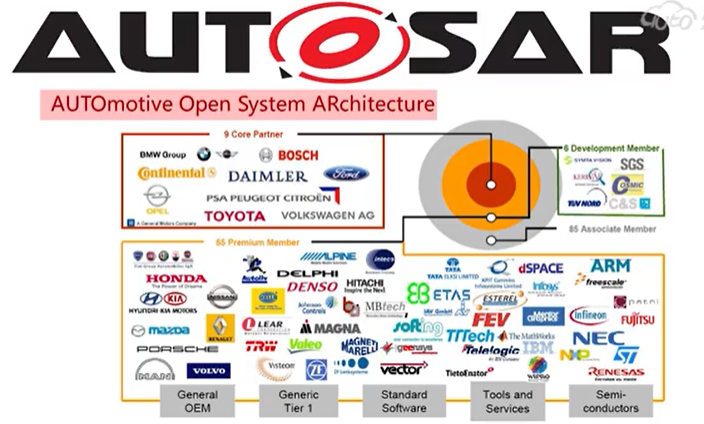
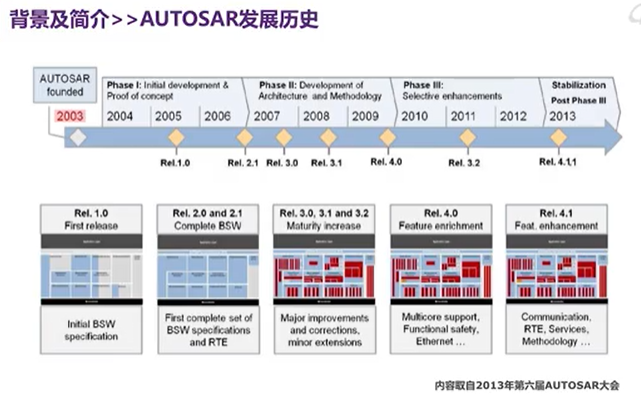
# AutoSAR架构介绍及方法论介绍



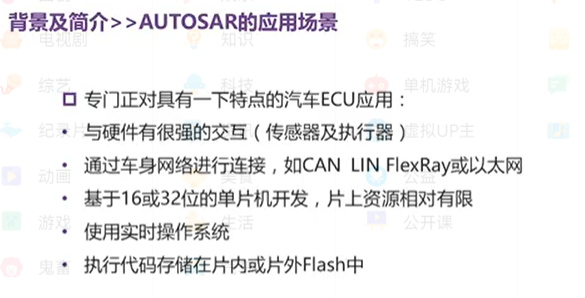
Autosar应用背景及介绍

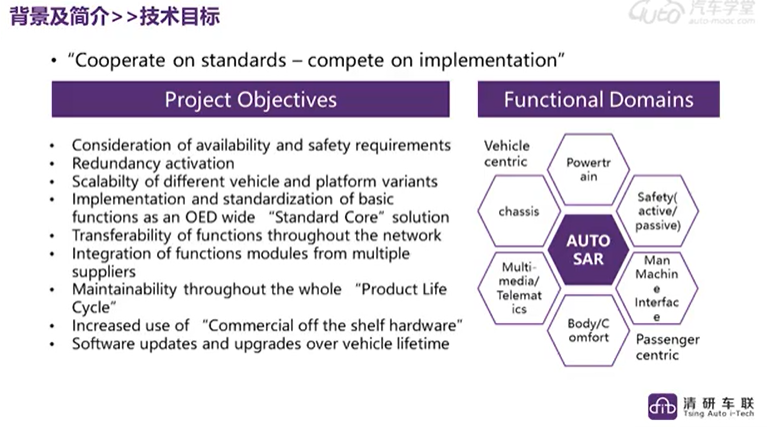


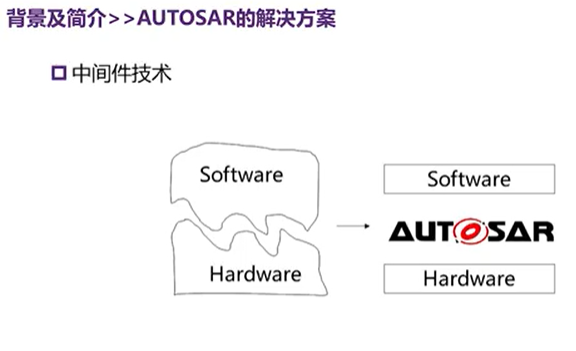




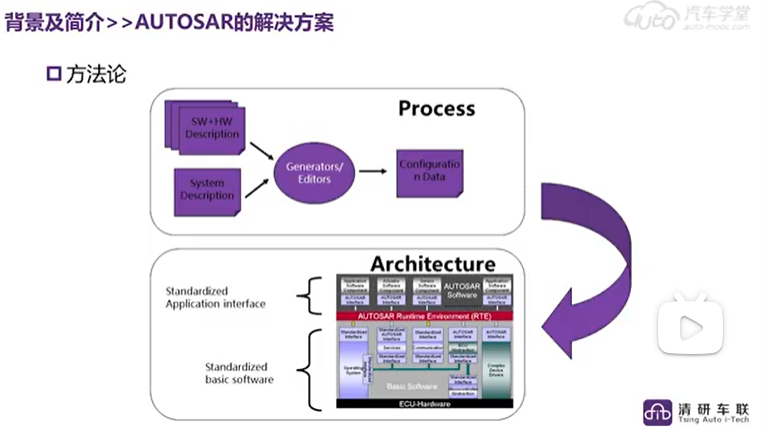
Autosar Adaptive platform,与原先的classic platform相比，Adaptive platform的主要区别是面向服务的。是surface origined archeteccher.服务可以驻留在本地的ECU上，也可以在远程的ECU上，通过网络通讯的形式，进行服务的调用。当前介绍的是Autosar classic platform的内容，程序还是主要驻留在本地的ECU上。



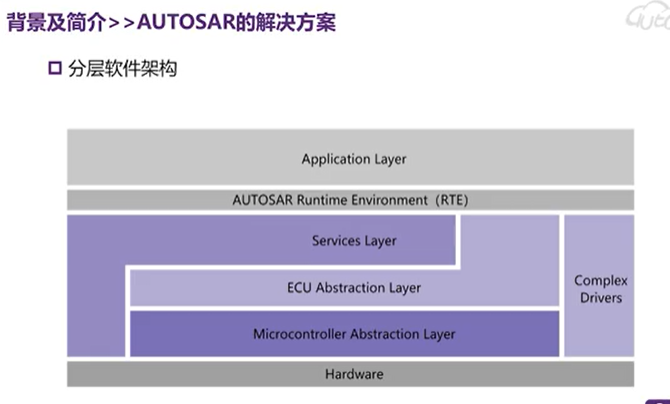


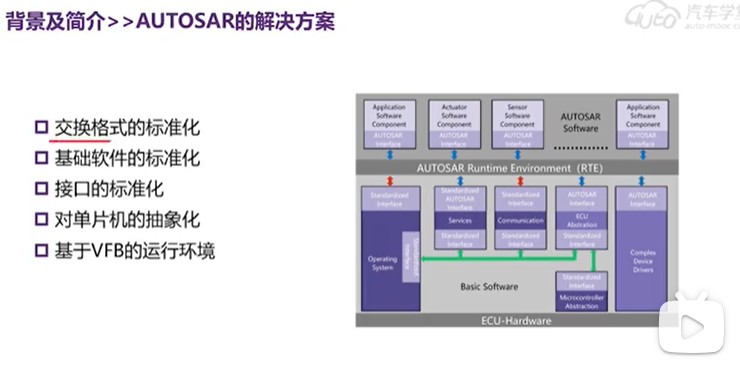


Autosar作为一个中间件，在使用了中间件之后，有了模块化和标准化的平台，使得我们在很多软件模块都是可以重用的，那么不需要去针对新增的功能，或者是一些变动，完全去对软件进行开发，那么因此我们使用了Autosar这个中间件实现了硬件和软件的解耦。Autosar尝试去统一以前的硬件和软件的开发平台，从而可以对开发的过程进行一个简化，为了实现这个目的，它把软件架构进行了标准化的定义，同时呢，还提出了一套标准化的开发方法。那么这个方法我们称为Autosar的方法论。

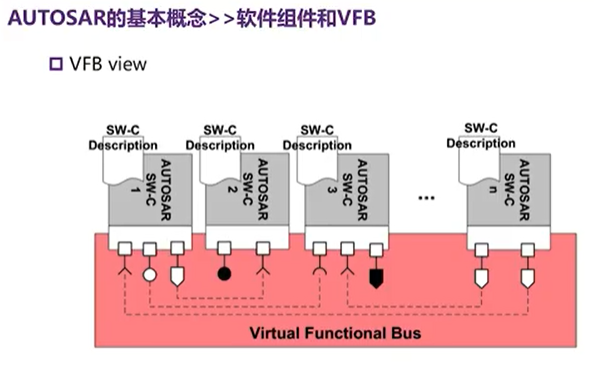


Autosar提出的标准化软件架构



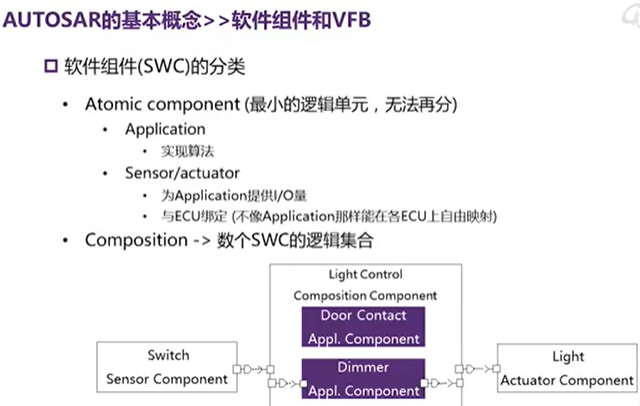


## Autosar的基本概念



VFB Virtual Function Bus,上图是一个非常抽象的对ECU功能进行描述的视图。在上图功能中可以看到，所有的swc（软件组件），都是通过Port连接到Virtual Functional Bus上。所有这些功能组件实际上都是对ECU功能的描述，最后在实现过程中，都会生成在ECU上执行的代码。这些功能组件，本身也是应用层软件。

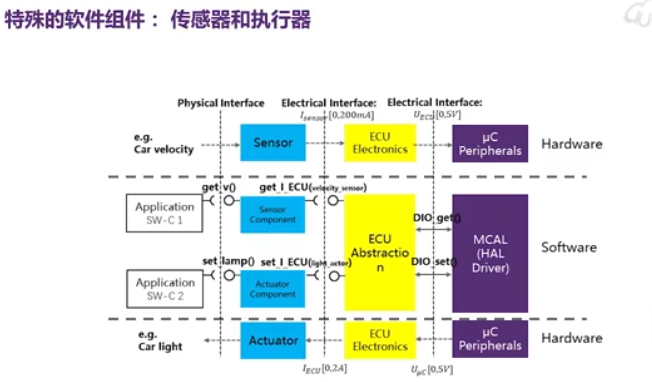
在开发之初并不关心它们在ECU上的具体位置，也不关心他们时通过什么样的方式来通讯，都是使用抽象的VFB来对它们进行连接。那么VFB可以进行SWC之间的数据交换，同时也可以提供SWC之间的服务的调用，它可以表示ECU内部的通讯，比如说内部总线通讯，同时呢也可以表示外部的一些总线通讯，比如CAN总线，LIN总线。那么只是在VFB view下，我们将所有底层的信息都进行了抽象化，使得应用层的开发可以完全独立于ECU的映射和ECU的一些物理上的特性，关注本身应用层软件功能的开发。



SWC是Autosar的最小逻辑单元。Application swc可以实现算法；Sensor/actuator swc 为Application提供传感器的输入，同时要让应用层的swc能够来对ECU进行操作。

在框的中间是一个light control的swc，包含2个软件组件，一个是对车门状态的判断，另外一个是调光器的控制；那么在swc两侧呢又包含了2个swc，在左侧是Switch Sensor Component是传感器的swc，在右侧是Actuator的swc。这样就完整组成了对灯光调节的功能。

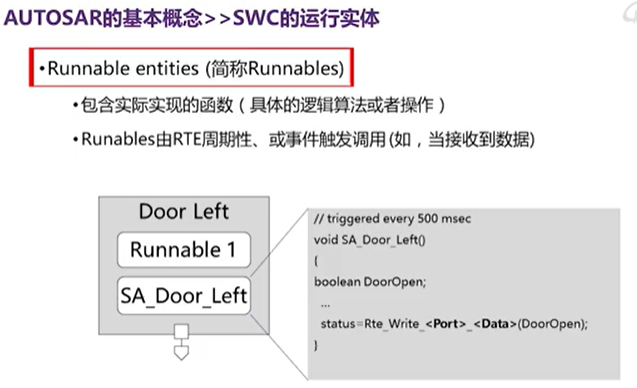
那么在这一页是我们对两类特殊的软件组件，传感器和执行器来进行描述。



同时上图也是我们Autosar软件和ECU硬件相互对应关系的描述，从上图中可以看到在图的最左边，有两个应用层的swc，swc1是可以对传感器进行读取的，所以可以看到在其右侧是一个Sensor的传感器，Sensor Component在硬件上对应的是硬件的传感器，硬件的传感器可能是一个开关量。那么在它的下一层呢是我们所谓的ECU Abstraction。ECU抽象层对应的是ECU上的一些硬件设计。再往下呢是Mcal单片机抽象层，它的功能是对单片机的外设来进行抽象。

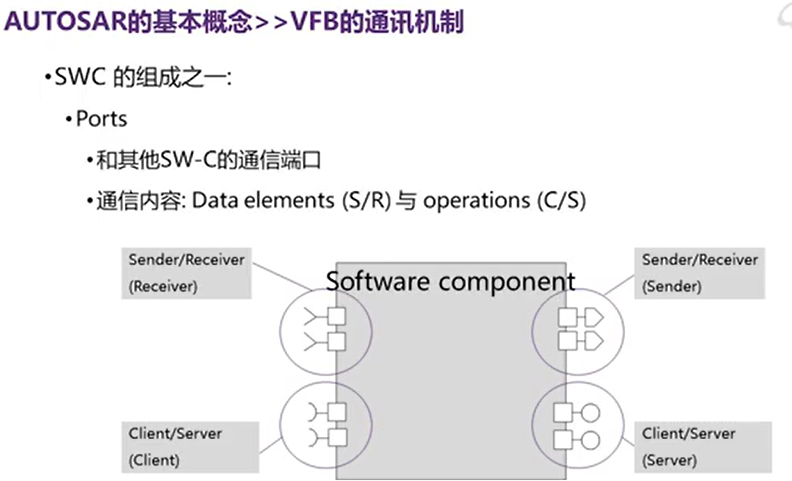
在swc2这部分，它是可以对灯光及进行调节，所以与它相连接的是一个Actuator的Component。Actuator的Component对应的硬件是执行器，可能是一个灯管的控制器，功率器件，与之对应的也是ECU的抽象层，ECU的抽象层将灯光调节的硬件电路，电频信号，IO口，这些信息也进行抽象；最下面的也同样是单片机的抽象层，可以理解为一个单片机的输出端口。通过单片机底层软件把硬件信息进行抽象。

这是2类swc以及软硬件对应关系的描述。



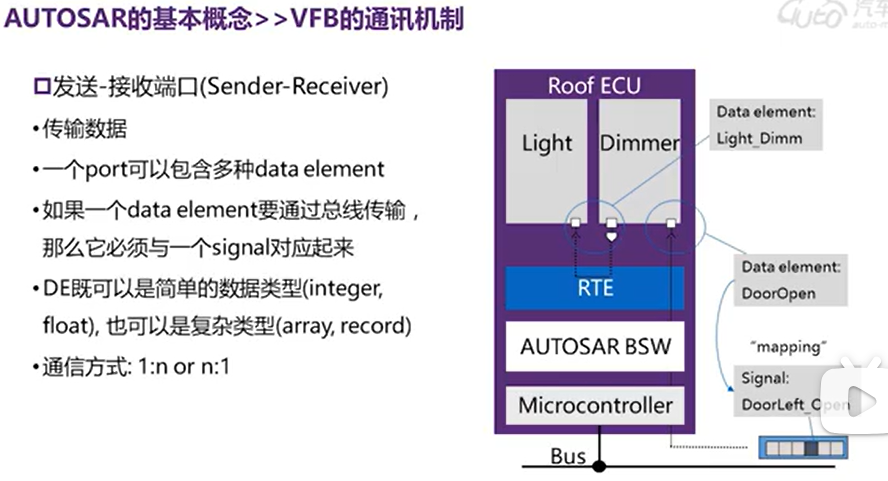
Swc的运行实体称为Runnable，Runnable从更加通俗的意义上来讲呢是，实际上是swc软件组件里的函数，是由具体的逻辑算法和操作来组成的，可以进行周期性或者事件触发的调度。这些函数组成了swc，又可以通过rte，一些runnable进行定时调用。把这些Runnable 通过rte映射到操作系统里以后，就可以实现rte的调用。在上图中可以看到，是一个Door Left，左边门的swc。这个swc里面包含了2个Runnable，其中一个Runnable叫做SA\_Door\_Left，在右侧的图里面可以看到对应的SA\_Door\_Left的函数。所以Runnable本身就是一个函数，那么在上面注释中可以看到是周期性进行触发的，那么就是说通过RTE把Runnable映射到了一个500ms里面的task里面，那么这个runnable就会被task进行定期的调用。

在前面的VFB view里面，还提到另外一点就是每个swc都有对应的port，这是port是swc与vfb总线交互的交互点，同时呢通过这些port，swc可以与其它的swc进行数据上的交互，或者是服务上的调用。



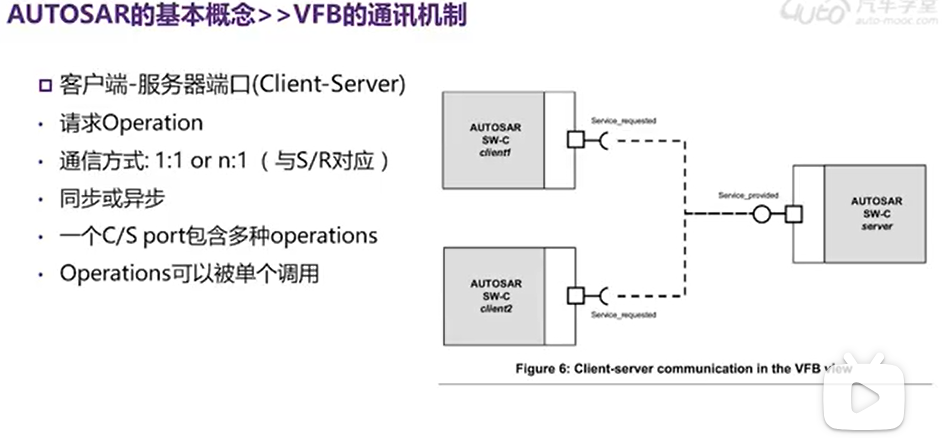
那么从上图可以看到swc的port呢，可以分为2类，一种是Sender/Receiver Port。这种port在Autosar里面是以直线或者五边形的形式来表示的，分为2类，sender是用来进行数据发送的，receiver呢是用来进行数据接收的。同时在Autosar里面，允许sender、receiver进行1对多的通讯，也就是说同一个port可以发送给多个receiver。那么另外一类端口，下面看到的这类端口，即Client、Server端口，一个用来提供服务，一个用来请求服务。在Autosar中，这两类port分别是用圆形和弧形来表示的，圆形是Server的port，弧形是Client的port。Client port可以通过vfb总线连接到Server的port上去。然后对Server能够提供的服务来进行调用。

由上例子可以看出sender和receiver的一些特点

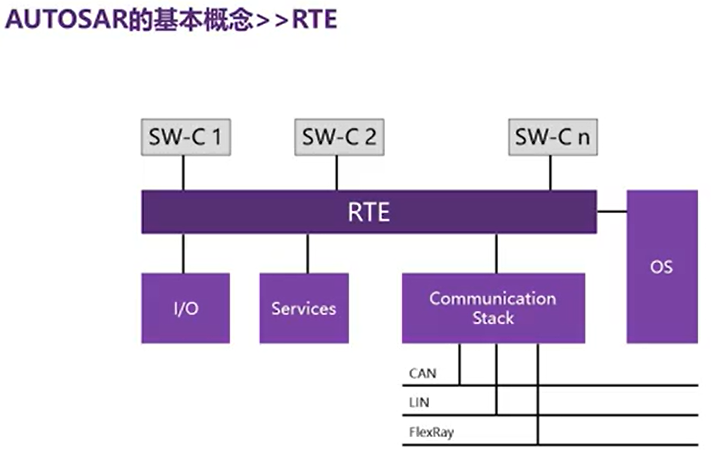


首先功能是进行数据传输，在port里面，包含了很多data elements，这些数据元素可以是不同的数据类型，可以是bit类型的，布尔类型的，也可以是global类型的，那么用这些data elements来组成port之后呢，就可以进行数据一对一或者数据1对多的传输。在上图例子中可以看到实现了2种数据通讯，一种是通过外部总线进行的数据通讯，可以理解为外部的CAN总线上发过来的一个信号，信号的名称叫DoorLeft\_Open, 只是左边门被打开了这样一个状态。那么通过sw层就把这样一个CAN信号，映射成一个data element，包含在data port中去。那么从Dimmer swc这个角度来讲呢，看到的是这样一个抽象的信号，并不关心信号是从外部总线来的还是从ECU内部总线来的。另外一个例子呢可以看到Data Elements是叫Light\_Dimm,可以理解为调光器给灯光控制器的swc进行数据通信的接口，那么这实际上及时内部的数据传输。从这个角度可以理解 rte或者vfb是把内部和外部的通讯都进行了抽象， 从应用层的角度来讲，这些抽象的数据通讯呢，他们都是一致的。

介绍完了sender和receiver之后，来介绍下Client Server



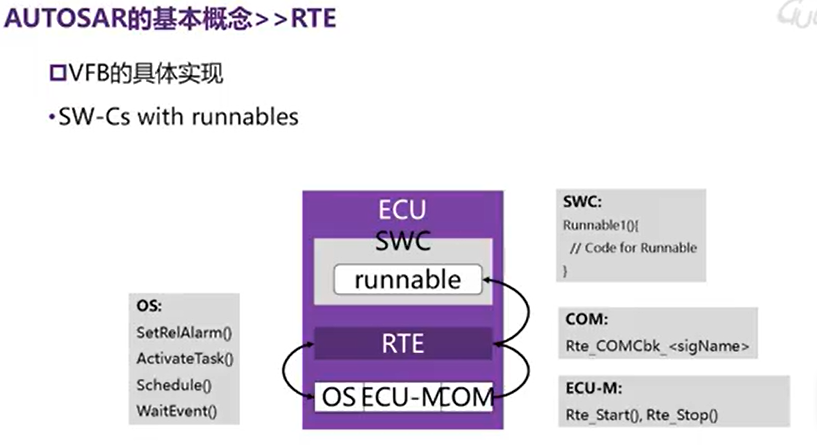
客户端和服务器的通讯接口，这个端口是用来请求Operation，通信方式是1:1或者1：n，请求方式可以是同步或者异步的。同步的意思是客户端会对服务进行等待，等到服务结束之后，才会退出Operation的请求；异步的通讯请求呢，Client不会对执行结果进行等待，请求完成之后就可以去执行其他的操作了。与sender、recerver一样，一个c/s port可以包含多种Operation，可以由多个Operation来组成，里面的Operation可以被单个调用。也就是说调用port之后，并不要求调用port里面所有的操作。



Rte是vfb总线在ECU上的实现，rte以上是vfb view，所有的swc通过rte总线进行相互连接，进行数据交互，进行他们的交互关系。那么在RTE以下就是BSW，基础软件层。在rte这一层实际上是将ECU的底层软件和ECU的硬件进行了隐藏。那么我们就不用关心具体的IO是什么样子，服务的提供是由哪个模块提供的，或者信号是通过什么样的总线协议来进行传输的。那么也就是说RTE的功能就是对底层软件的抽象化。Rte实际上有2方面的功能，一方面是实现数据的通讯，另外一方面是实现任务调度。刚刚讲到的swc是由很多Runnable来组成的，那么这些Runnable在时间上需要进行调度，需要进行运行的话，就需要对RTE进行配置，需要通过RTE把这些不同的Runnable映射到不同得OS task上。



再同时，RTE产生对应的事件来触发这些Runnable来运行，在运行的时候，实际上就是OS Task来调用代码。比如前面例子中Runnable代码，这些不同的代码就会被归集到不同的例子中去，再由Task来进行执行调度。同时RTE还需要对OS里的信息来进行一些配置，比如tasks，events时间，alarm时钟等。RTE因为是VFB总线的实现，所以和每个ECU有关系，所以需要针对不同的ECU来进行定制，这样RTE就对OS操作系统进行了抽象，同时也防止SWC直接访问OS和BSW。

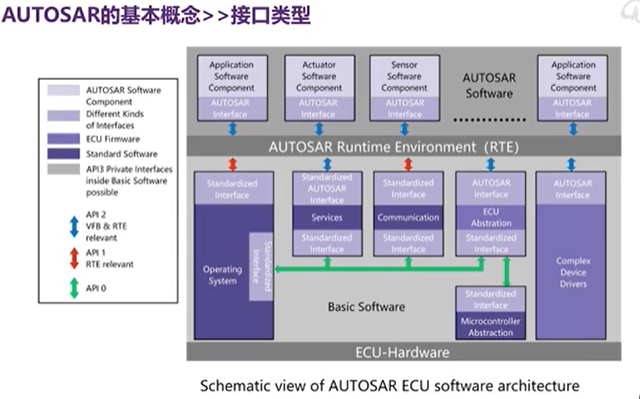


可以通过RTE的触发事件来对Runnable进行调用，这里有几类触发事件。第一类SWC对函数的调用，sender/receiver，client/server port对函数的调用；第二类试ECU-M, ECU-M试ECU Manager模块，它可以调用RTE start、RTE stop这样的函数来改变ECU的状态同时也可以产生对应的rte事件；第三个可以通过回调函数COM来进行数据的调度；第四个是可以生成OS的事件，通过操作系统的事件，再对RTE映射里的任务里的Runnable进行调用。

接下里介绍RTE的数据交互



RTE最重要的功能是对swc之间、或者swc和bsw之间的通讯的机制。前面讲了RTE是VFB View的实现，所以所说的sender、receiver之间的port，或者client server之间的port或者ECU内部的通讯，ECU之间的通讯，以及一些COM回调函数的实现，都是在RTE层面进行实现。除了数据交互之外，RTE还有其它的一些特性，可以对数据进行校验，保证数据的一致性，同时可以支持不同的数据格式，比如简单的数据格式，基本的数据格式，也可以支持Record结构体这样的数据格式，另外在RTE里面还可以实现swc的多副本，同一个swc可以进行复制，就生成了相同类型的不同swc。



Autosar里面定义了3种类型的接口

标准接口是使用特定语言进行定义的标准化函数，通常是用在ECU内部的函数通讯，使用标准化的接口就不能实现在网络上的通讯了，因为不符合Autosar接口标准；

与之相对的是Autosar接口，它是用来定义软件swc模块和swc与bsw通讯的端口，那么由上图可以看到，在swc这一端的接口，都是Autosar的interface。同时还有一些负责服务，负责通讯的模块，他们也是符合autosar的interface。在总线上进行通讯。

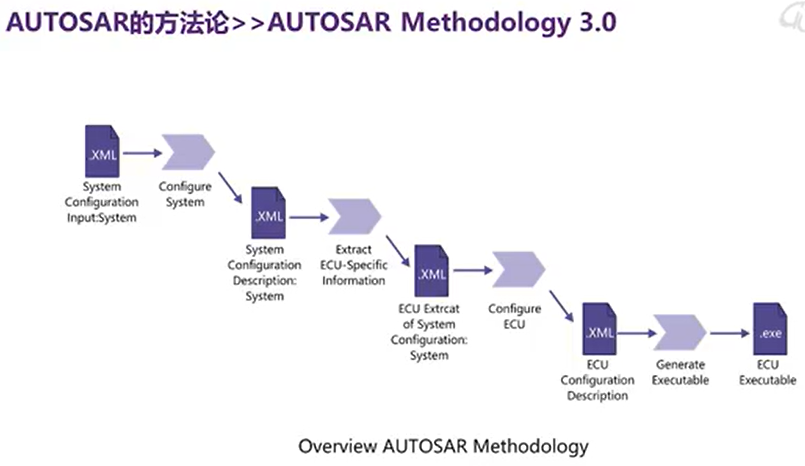
除此之外，还有一类叫做标准化的Autosar的接口，这是在Autosar接口的基础上，对语法的定义或者语义的定义进行了标准化的Autosar接口，通常是用来提供标准化的Autosar接口的。所在上图可以看到，在service layer，有一些端口是所谓的标准Autosar接口。

## Autosar方法论

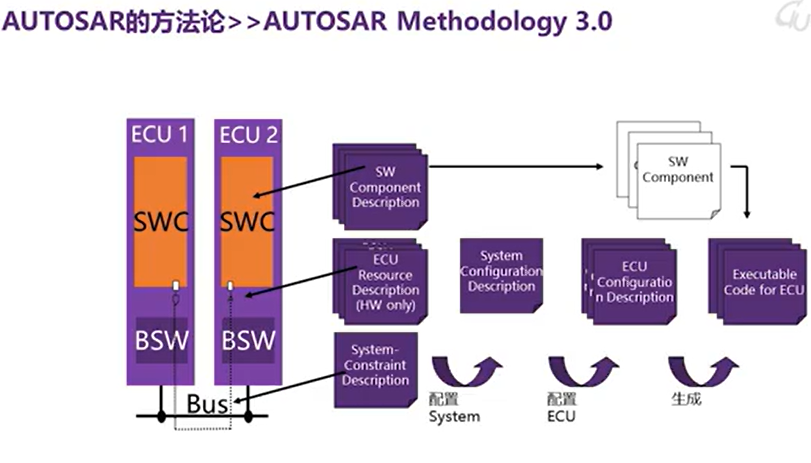


在Autosar中，定义了很多信息，包括开发过程中的角色以及分工，在每个阶段使用什么样的工具进行什么样的工作。工作的输入是什么，输出是什么，这就是Autosar方法论的内容。

Autosar方法论中使用了结构化的开发方法，可以在开发最开始的时候，对需求上的一些缺陷，比如功能上要求的一些权限进行识别。再最开始的时候，就可以对这些缺陷进行修正而不是在软件开发到最后的阶段，比如最初定义的系统上面有什么问题。这即是Autosar方法论。

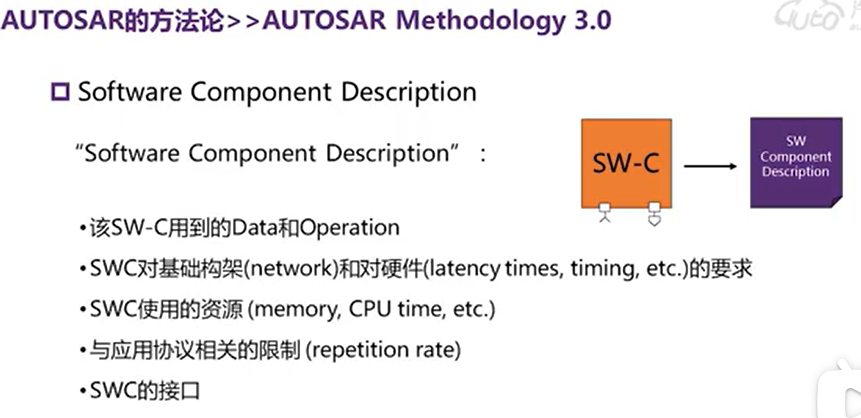


上图是Autosar方法论3.0的概览，可以看到概览中从系统阶段到ECU的定义阶段，再到软件的开发阶段，最后生成ECU的执行代码。实际上通俗来说，这是汽车电子V流程的一半。



开发过程是由3类描述文件开始的，描述文件包括swc的描述文件，ECU的资源文件，和系统限制的描述文件。这些都是输入，通过输入，可以对这个系统来进行配置，这个阶段产生的是system的配置描述文件。在系统配置文件完成之后，要对ECU来进行配置，要根据ECU具体任务的划分，来进行ECU的配置。这个阶段的产物就是ECU的配置文件，最后再由ECU的描述配置文件加上swc代码，最后集成生成ECU上可以执行的代码。所以可以通过这样一个流程去对AUtosar进行规范。同时配置过程是可以迭代的，如果在哪个阶段发现前面的设计是有问题的，是可以再次进行修改的。另外所有的组件，描述文件，都是有统一的格式的，比如xml文件。AUtosar有很多模版，可以在模版上面进行ECU的开发。

接下来对方法论的几个阶段进行分别的描述



首先是Software Component Description，这是软件组件的描述文件。软件组件描述文件包含了软件组件相关的一些信息，比如用到了什么样的Data和version。这即是后面要配置swc要用到的port信息。

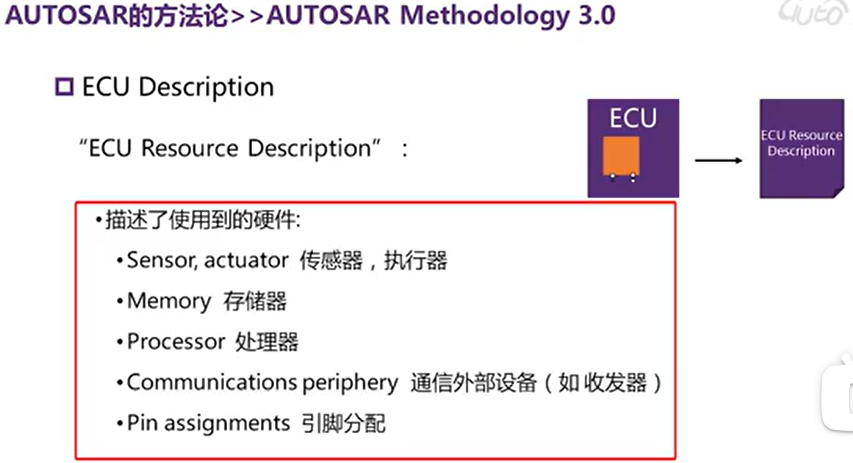
第二个是swc对网络架构和硬件的要求，即信号可能从什么样的总线上获得的，同时对硬件的执行时间或者其他的一些方面，有什么样的要求。

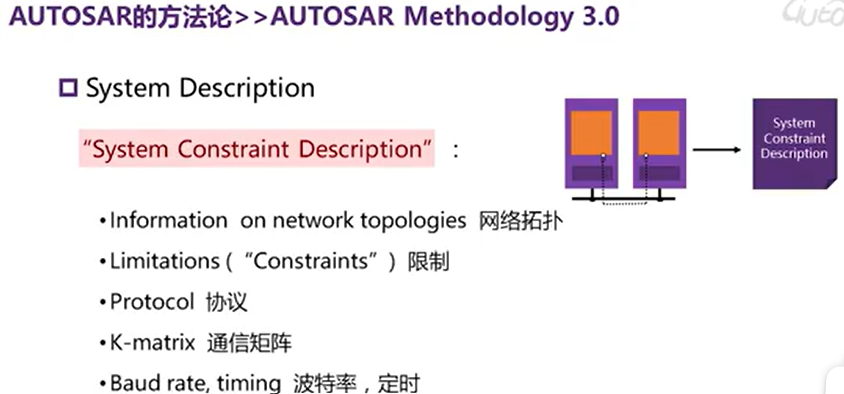
第三个是它所要使用的资源，需要占用多大的memory，需要使用多少的CPU来执行swc。

第四个是应用协议相关的一些限制，比如swc多长时间可以进行调度，

还有就是swc的接口，以上是swc description中需要描述的内容

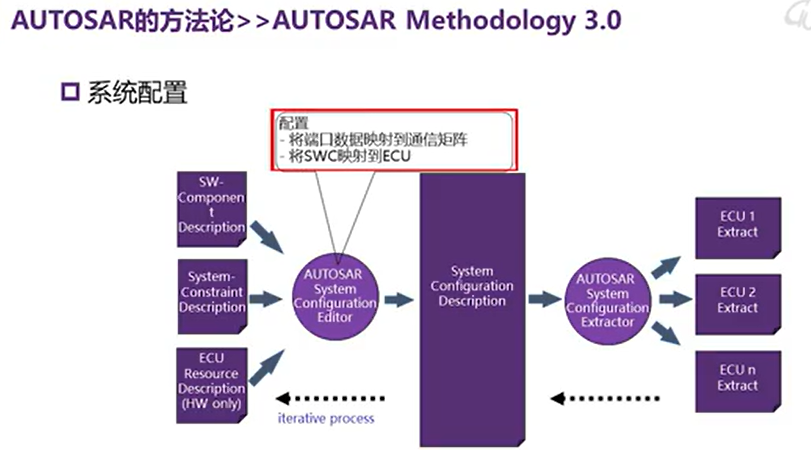
第二部分是ECU resource 的description，这是描述了可用的硬件资源，包含了ECU上有什么样的Sensor，有什么样的执行器，有多少memory可用，使用的是什么样的处理器，它们使用的外部通讯设备是什么样的，包括引脚分配是什么样子的。也就是说是对ECU硬件的一个描述文件。通过这些描述文件，可以知道ECU有多少资源是可以通过软件使用的。也决定了后面的哪些功能会分配到哪些ECU上面。





还有就是System Constrain Description，这是系统约束的一些描述文件。简单来讲就是整车电子电器的架构。这里包括了网络拓扑，通讯限制，使用了什么样的通信协议，通信矩阵是什么样子的，波特率，频率，时间间隔是什么样子的。这是系统限制的描述文件。

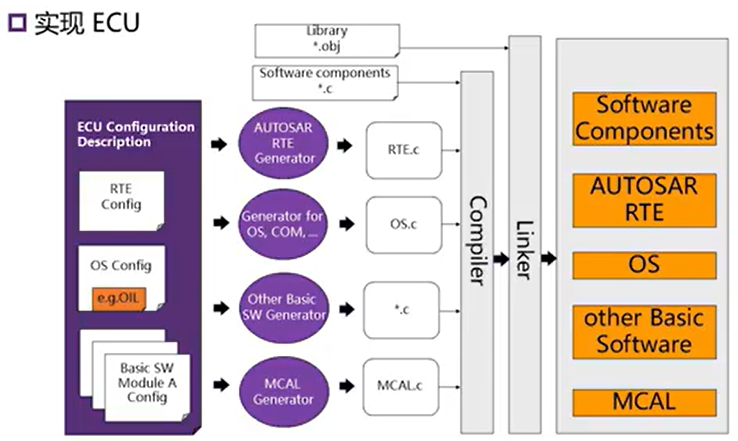
根据前面说到的3个描述文件，swc描述文件，系统限制描述文件和ECU资源描述文件。可以将对应的信息进行配置，生成系统配置文件。



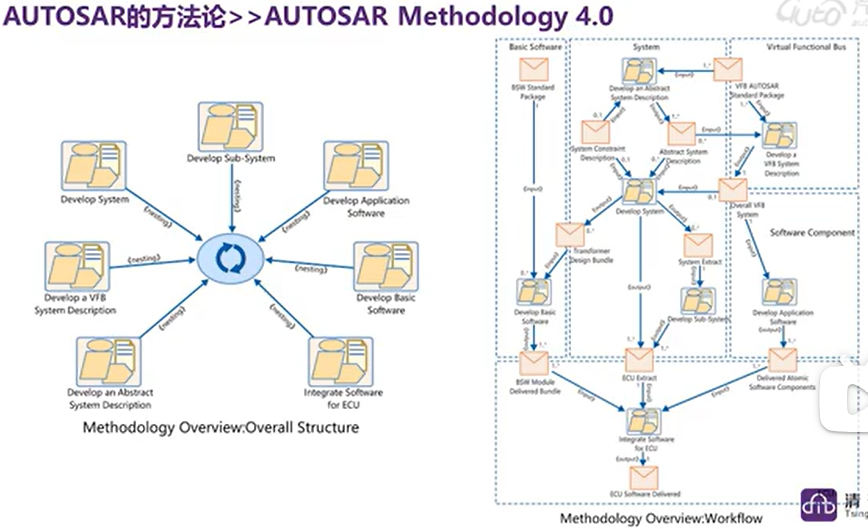
配置文件里可能包含swc，对ECU的映射，还有包含了数据端口到通信矩阵的映射。Swc到ECU的映射即swc需要划分到哪个ECU上并进行实现。端口数据的映射就是说端口数据，是来自哪个总线，或者是来自ECU内部的通讯接口，这即是系统配置过程。



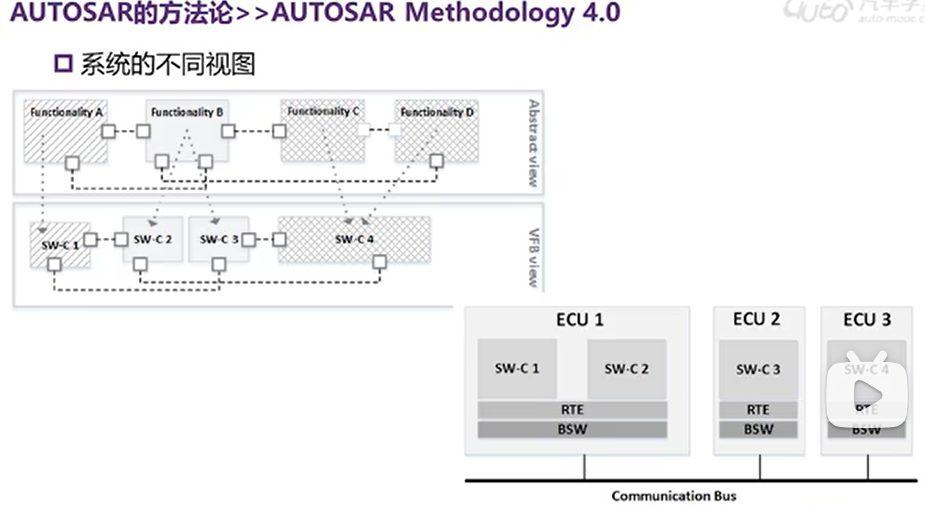
接下来则是ECU配置过程，在此过程需要将Runnable映射到对应的Task中去，还有生成对应的一些描述文件。这里可以看到有swc实现的list，rte的描述文件，rte的配置文件，操作系统的配置文件，还有bsw的配置文件。实际开发过程中，这即是ECU的开发过程。根据这些配置文件，可以使用对应的代码生成工具，就可以生成相关的ECU这些代码了。



最后是ECU的实现，根据前面ECU的配置文件以及对应的代码生成工具，比如rte的代码生成工具，或者操作系统生成工具，还有底层软件的生成工具。这些生成工具都会把对应的配置文件转化为对应的C代码，这些C代码，再配合上我们使用的AUtosar的library文件，以及swc实现的文件，最后把这些所有文件进行整合。然后可以生成ECU可以执行的二进制代码了，这就实现了ECU软件的开发过程了。



AUtosar4.0对开发过程进行了进一步的完善，上图左边是方法论的overview，是整体架构。右边是开发的流程，从整体的结构上面我们可以看到，它是在我们的系统设计之前，又增加了2个部分，一个叫abstract System的开发过程，另外一个是vfb System的开发过程，这是系统开发的进一步抽象，在更高层次上进行系统的开发。这里是把开发流程划分为了3块，这三块分别是bsw的开发，系统的开发，和vfb的开发，也是对原先AUtosar 3.0的扩充。



从图中可以看到3种视图，最上面是abstract view，这里所有的模块都被称作为功能模块，这里还没有所谓的swc的概念。这些功能块有相互的交互相互的接口，这是非常抽象的过程。再到vfb view以后，这些功能块就被进一步的映射或者拆分，对应的swc，成为软件组件模块。软件组件模块之间相互沟通是通过vfb总线来实现的，所以叫vfb view。再往下就是系统视图，即ECU视图，可以看到swc1和swc2就会划分到了ECU1中去，也就是说功能是由ECU1这样的硬件进行承载的和实现。对于对应的通讯也被映射到了它们使用的总线上去，这即是autosar不同的系统施图。

## Autosar分层软件架构

