Document Title	Guide to Multi-Core Systems
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	631
Document Classification	Auxiliary

Document Version	1.1.0
Document Status	Final
Part of Release	4.1
Revision	3

Document Change History

Date	Version	Changed by	Change Description
31.03.2014	1.1.0	AUTOSAR Release Management	Clarified terms
18.01.2013	1.0.0	AUTOSAR Administration	Initial release

Table of Contents

1 Introduction

This document is a general introduction to the development and configuration of AUTOSAR-compliant software for multi-core systems. As of release 4.1.1, it addresses the allocation of AUTOSAR BSW modules [1] to partitions on multi-core systems and their interaction only. The allocation of BSW modules to different BSW partitions allows for both enhanced functional safety and increased performance.

This document contains the description of multi-core extensions to BSW in AUTOSAR 4.1.1, which addresses performance use cases and is limited to backward compatible changes. Moreover chapter 6 of this document contains information about future work we expect for upcoming, not backward compatible releases. Information on multi-core aspects that are not specific to the BSW, but also affect application software, are planned to be contained in future versions of this guide.

This document is outlined as follows: Chapter 2 gives an introduction and a technical overview of distributing the BSW across different partitions and cores. Chapter 3 addresses developers of BSW modules and explains how to implement modules that can be run in multiple partitions. Some specific extensions to the BSW scheduler, which may have to implement additional methods for the parallel execution of BSW modules, are presented in chapter 4. Chapter 5 describes the configuration aspects related to the parallel execution, and chapter 6 provides an outlook to upcoming concepts for the parallelization of the BSW, which can be expected in future versions of AUTOSAR that are not backward compatible. A glossary of technical terms and a list of references to external information are provided in chapters 7 and 8.

2 Overview

This chapter contains a description of the supported scenarios for distributed execution of BSW modules on several partitions and cores and a number of use cases in which a distribution of the BSW can enhance performance. It also introduces basic synchronization concepts applicable to distributed BSW execution, and an introduction to inter-partition communication.

2.1 Supported Scenarios

It is possible to assign functional clusters of BSW modules (“"BSW Functional cluster"), which are used by applications to access buses, non-volatile memory, I/O channels, and watchdogs, to different BSW partitions for safety or performance reasons. The clustering of BSW modules is currently not standardized. Parallel usage of the same type of functional clusters in different partitions ("duplication") is not generally supported, but it is possible by using a master satellite approach.
Functional clusters to partitions may be assigned such that

- a BSW functional cluster is only available in one partition
- a BSW functional cluster is available on all partitions with all interfaces
- a BSW functional cluster is distributed over multiple partitions, possibly with partition specific subsets of functionality, to allow a high grade of concurrency.

In either of these scenarios, the following restrictions apply:

- There is currently at most one BSW partition per core. (This may be subject to change in future AUTOSAR releases, cf. chapter 6.)
- All partitions that contain BSW modules are trusted.
-

With the aforementioned restrictions, AUTOSAR supports the scenarios listed above. In doing so, it addresses the following essential features:

- All code for communication between BSW partitions can be generated for automatic adaptation to different system configurations. The cross partition communication mechanism can be generated with focus on efficiency, or, in future releases to help to provide freedom of interference.
- If access to system services (which are not part of a BSW functional cluster) is required, efficient access from each BSW partition that needs the system service is supported.
- Efficient access to HW abstraction and drivers is supported in each BSW partition, if required.

In all scenarios, the communication between different module entities remains unchanged (in comparison to BSW running in a single partition).

2.2 Performance Use Cases and Hardware Assigned to Different Cores

The following use cases are examples for how system performance can be improved by allocation of the BSW to multiple partitions and cores, and how systems where the access to the peripheral hardware is assigned to multiple cores benefit from the allocation of the BSW to multiple partitions and cores.

- To increase system performance and to reduce resource consumption in systems that are distributed over several cores, it may be necessary to allocate functional clusters of BSW modules to different cores, e.g. communication modules on BSW partition "A" and I/O modules on BSW partition "B", depending on hardware architecture, load balancing and on distribution of SW-Cs. In particular, if HW resources are accessed exclusively by one core in a Multi-Core system, the performance is increased by locating the corresponding BSW users, services and drivers on that core.
- Signal gateway functionality is implemented by allocating a FlexRay cluster on one core and a CAN cluster on a different core. The two COM modules need to be synchronized in this case, and there must be some direct cross core communication between the two COM instances. One of the COM modules might be the master COM that coordinates the satellite COM on the other core.
- Two communication clusters are located on different cores, one accessing a CAN bus and the other one controlling a FlexRay bus. In case the application SW located above one of the communication clusters on the same core needs to send on both buses, the core local COM modules can directly communicate with their counterparts on the other core, to efficiently send the signal over either CAN or FlexRay. For received messages, COM has no information about receivers above the RTE. Therefore, COM has to forward the signals on the receiving side to the RTE, and the RTE is responsible for communication.

2.3 Technical Overview

Below is a short summary of the technical solution as described in the following sections:

- Define clusters of BSW modules that contain preferably all three layers of a stack, or, if needed, a subset of modules of a stack (e.g. communication, memory, I/O stack).
- Module entities can be split into a master and satellites, which are assigned to different BSW partitions. Masters and satellites can use non-standardized AUTOSAR interfaces, for internal cross partition communication. The master/satellite approach is mainly used by distributed system service modules and for communication between BSW clusters of the same type.

The proposed solution meets the demands on performance and safety while minimizing the impact on already standardized BSW module interfaces (RS_BRF_00206, RS_BRF_01160). Most changes are hidden within modules (e.g. by providing master/satellite implementations) without affecting other modules. Interfaces between different modules do not change.

2.3.1 BSW Functional Clusters

BSW functional clusters are groups of functionally coherent BSW modules. Each functional cluster includes a set of BSW modules. It is possible to have several BSW functional clusters of the same type (e.g. several I/O clusters in different BSW partitions), each using a different set of modules (e.g. IOHWA + ADC in one partition and IOHWA + ADC + DIO in the second partition).

The following types of clusters might be standardized in a later release:

- Communication cluster
- Memory cluster
- I/O cluster
- Watchdog cluster

The allocation of BSW functional clusters to BSW partitions is determined by the usage of BSW modules by the application software. Functional clusters can be allocated to different BSW partitions, and functional clusters of the same type can be available in several BSW partitions. Different functional clusters can be allocated to the same or to different BSW partitions.

The same functional cluster can only exist at most once per BSW partition.

BSW functional clusters are used by applications or other BSW modules to access buses, memory, I/O channels and watchdogs, and they are usually required in one or few BSW partitions only.

The introduction of BSW functional clusters does not change the existing AUTOSAR R4.0 interfaces between the BSW and the RTE, which are mainly used to implement AUTOSAR services, i.e. to communicate with the application layer. It may however change the availability of standardized AUTOSAR interfaces on different partitions.

The internal structure of a BSW functional cluster, including its internal communication between BSW modules, and the communication with system services that the BSW functional cluster uses is not necessarily affected by the parallelization of the BSW, and it does not need to change. It may however be adapted, for example in order to fulfill special demands on concurrency like the support of different entities of the same module running in different partitions.

The communication and synchronization between modules in BSW functional clusters of the same type (e.g. in two communication clusters to support a gateway functionality) is not standardized. It will be implemented by communication between entities (e.g. by a master and satellites) of specific modules, which can use non-standardized interfaces for communication across BSW partition boundaries, see Figure 1.

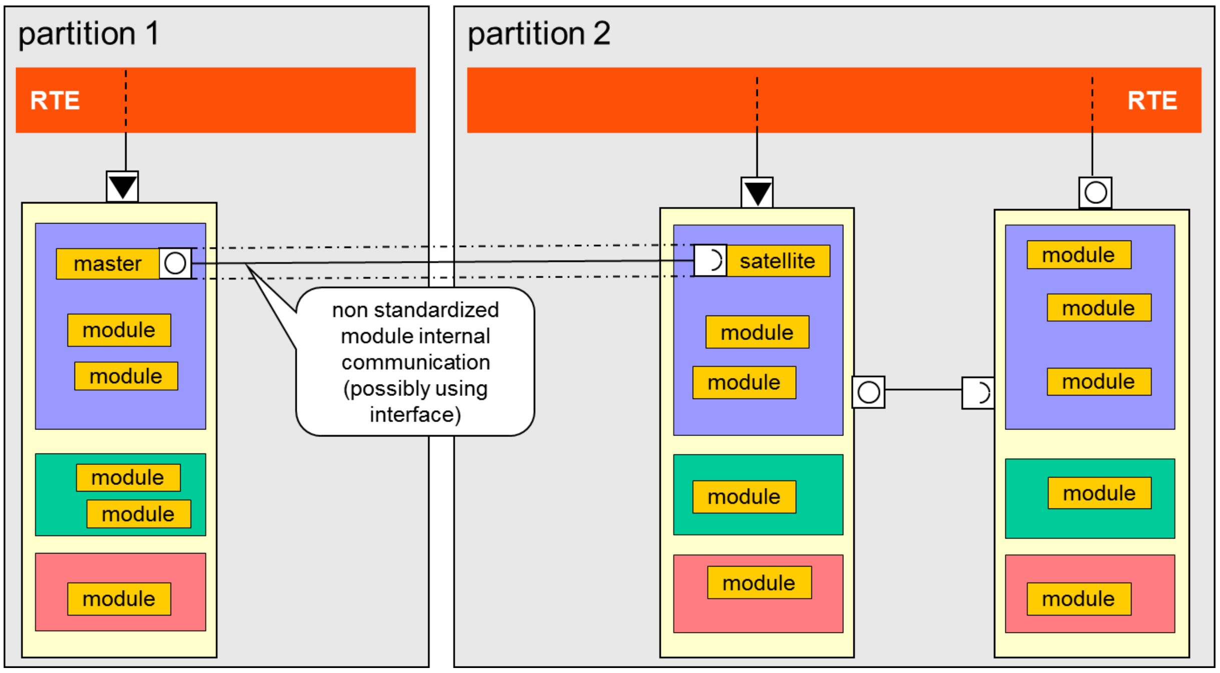

Figure 1: Functional clusters of the same type

Modules that do not belong to BSW functional clusters (some MCAL modules, system services) will always be accessed within the same BSW partition where the BSW functional cluster is located. As the interfaces do not change, these modules must be locally available in each BSW partition, if needed.

2.3.2 Inter-BSW-partition communication

Function calls to tasks that are supposed to be executed in a different BSW partition/on a different core cannot be implemented as simple C calls to this function, because these calls would be handled on the local BSW partition.

The BSW Scheduler (SchM) therefore provides functions to invoke masters or satellites of the same module on different BSW partitions using either client-server or sender-receiver communication. Details on this API of the SchM are explained in Section 3.3.

2.3.3 Determining the Partition for Service Execution

The actual BSW partition for the handling of an RTE event is determined by its task mapping. Basically, if an event is mapped to a task, it is executed within the partition assigned to this task. If an event is not mapped to a task, it is executed within the same partition as the task that caused the event. Details on the task mapping are described in Section 5.1 of this document.

Calls from BSW entities to other BSW entities are not mapped to a partition. They are executed wherever they are called. Therefore, several calls to a BSW function may be processed in parallel on different partitions and cores. Consequently such functions must be designed and implemented carefully w.r.t. parallel execution in different partitions; if necessary, they shall be reentrant or concurrency safe.

2.3.4 BSW partitions

Only partitions that have the configuration parameter *EcucPartitionBswModuleExecution* set to true can execute BSW modules. Such partitions are called BSW partitions. BSW partitions may additionally contain application software components above the RTE.

3 Parallel Execution of BSW modules

This is the chapter for developers of BSW modules.

3.1 Core-Dependent Branching

Because entities of the same module share the same implementation, even if they are running on different cores, different behavior cannot be realized by different code. Instead, the specific behavior shall be determined by runtime information. It is possible for example to use the core id for this, i.e. branch the control flow depending on the return value of the OS APIs `GetCoreID()`, or also `GetApplicationID()`.

As an alternative, the BSW partition specific Service SWCs can invoke different Runnables in the BSW modules.

3.2 Master/Satellite-approach

Modules that need to be accessed in different BSW partitions can be implemented using the master/satellite pattern.

The distribution of work between master and satellite is implementation specific. One extreme is that the satellite only provides the interfaces to the other modules in the same BSW partition, and that it routes all requests to the master and answers back to the other modules. At the other extreme, the satellite can provide the full functionality locally (e.g. local mode management for a complete application which runs in the same BSW partition) and only synchronizes its internal states with the master, if necessary. There might even be several masters for different functionality, e.g. two PduR masters for a distributed PduR gateway.

The master coordinates requests from the satellites and can filter or monitor incoming satellite requests. The master and one or several satellites are treated like being one module entity in some respect:

- Master and satellites are always vendor specific solutions, coming from the same vendor.
- The interfaces of master and satellite to other module entities in general are the same as specified in AUTOSAR R4.0 for traditional modules. Master and satellite should provide the same APIs. This means that when migrating to partitioned systems, existing module entities can be replaced by a master and one or several satellites, in most cases without changing other modules. Exceptions might be module internal adaptations to additional delays which are caused by inter-partition communication.
- Master and satellites may have the same entry points in each BSW partition (i.e. they start executing the same functions from shared memory) and internally branch (e.g. by using the "GetApplicationID ()" API) to master or satellite specific code according to the OS-Application (partition) they run in. Depending on the build strategy, other implementations might be possible in multi-Core systems if each core can execute its own code. Also, satellites might share the same code without further branching.
- Master and satellites may have different entry points with different APIs in each BSW partition to allow a high degree of concurrency.

- The communication between master and satellites is not standardized. It is considered to be module-internal and is not visible to other modules.
- The communication between master and satellite can be initiated in either direction (i.e. by both the master and the satellites), as well as from one satellite to another one.
- All interfaces between masters and satellites are only allowed to be connected within the same distributed module.
- The communication between master and satellites can be implemented within one BswModuleEntity, or between different BswModuleEntities that belong to the same BSW module.
- Depending on the application, usage of master/satellite may be appropriate or not. For example, it may be more efficient to use separate, partition specific watchdog clusters, which work independently from each other, rather than using the Watchdog Manager in a master/satellite approach.
- The master is the part of a distributed BSW module that coordinates requests by satellites and can filter or monitor incoming satellite requests. This may result in additional fault detection or fault mitigation mechanisms. Generally, all errors caused by distributed execution of a module should be handled module internally.

The master/satellite implementation is the standard solution for system services in partitioned systems.

Specific drivers also might have to provide local satellites, if the hardware can only be accessed from a different core. The standard solution, if possible, is to execute the same multi-core reentrant function in each partition and to separate the data to work on into disjoint sets, one for each partition. For example, the COM module may work on all IPDUs assigned to the bus that the BSW functional cluster of this module belongs to. Concurrent access to the same hardware or shared data needs to be protected, e.g. by ExclusiveAreas in this case.

In specific cases, modules within BSW functional clusters also need to be implemented as master/satellite, if the BSW functional clusters are duplicated and the entities in different BSW partitions need to be synchronized or need to exchange data. This might apply to the Watchdog Manager, the NVRAM manager, and to network and state managers in duplicated communication clusters. COM modules also might need to have a master and a satellite to implement cross partition gateway functionality.

3.3 Using the BSW Scheduler for Inter-Partition-Communication

The BSW Scheduler (SchM) provides a number of functions to support communication between BSW module entities that are executed in parallel. More precisely, it provides the following methods to handle synchronous and asynchronous calls (including callbacks) as well as sender-receiver communication.

The functionality is generally similar to that of function calls between SWCs and the BSW. However, because the RTE may not be available at certain points of time (especially during startup of an ECU), this functionality must be available within the BSW itself.

- `Std_ReturnType SchM_Call_<bsnp>[_<vi>_<ai>]_<name>(`
  `[OUT <typeOfReturnValue> returnValue]`
  `[IN|IN/OUT\|OUT]<data_1> ... [IN|IN/OUT|OUT] <data_n>)`
  or
  `Std_ReturnType SchM_Call_<bsnp>[_<vi>_<ai>]_<name>(`
  `[IN|IN/OUT\|OUT]<data_1> ... [IN|IN/OUT|OUT] <data_n>)`

  Invoke a client-server-operation, possibly crossing partition boundaries. The actual parameters `data_1 ... data_n` are information that is passed [IN] and/or re-passed [IN/OUT | OUT] to/from the called service.

  The presence of the parameter `returnValue` and its type `<typeOfReturnValue>` depend on the called service. For synchronous calls, the parameter is present and `<typeOfReturnValue>` is the type returned by the called service. For asynchronous client-server-operations and operations with return type `void`, the parameter is omitted.

- `Std_ReturnType SchM_Result_<bsnp>[_<vi>_<ai>]_<name>(`
  `[IN|IN/OUT|OUT]<data_1> ... [IN|IN/OUT|OUT] <data_n>)`

  Callback from an asynchronous client-server-operation, possibly crossing partition boundaries.

  The receiver of a callback is determined by the AsynchronousServerCallResultPoint of this callback. The AsynchronousServerCallResultPoint refers to the originating AsynchronousServerCallPoint, which in turn “knows” the calling module entity.

- `Std_ReturnType SchM_Send_<bsnp>[_<vi>_<ai>]_<name>(IN <data>)`

  Write data to a sender-receiver link between BSW modules, possibly crossing partition boundaries.

- `Std_ReturnType SchM_Receive_<bsnp>[_<vi>_<ai>]_<name>(OUT <data>)`

  Read data from a sender-receiver link between BSW modules, possibly crossing partition boundaries.

3.4 Using Shared Buffers (in systems without memory protection)

In systems without memory protection between the BSW partitions, system services and all BswCalledEntities can be called directly in every partition, including the complete call tree. This requires a reentrant, concurrency safe implementation.

The services and other called entities might work on module internal data, which is shared between different entities of the same module. All access to such data must be protected by ExclusiveAreas. Appropriateness of concrete protection mechanisms depends on the possible kinds of access. For example, concurrent writing generally needs to be prohibited, whereas concurrent reading may be acceptable, as long as only one partition writes at the same time.

BswSchedulableEntities are located on one core only and process the data periodically or event driven.

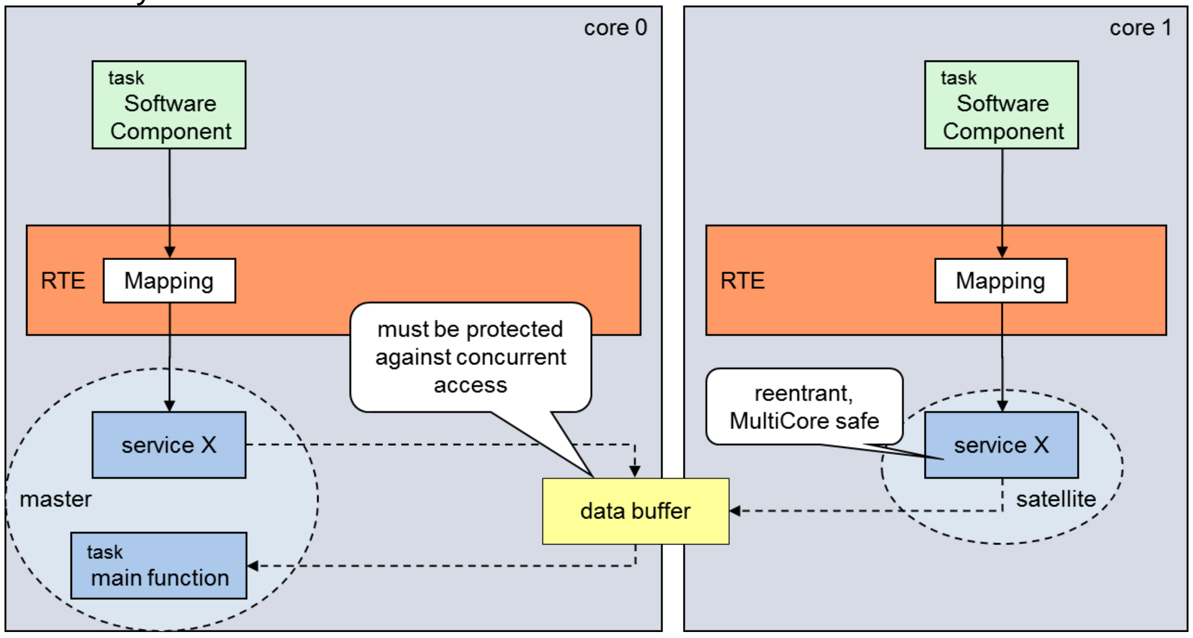

Figure 2: Invocation of same service on different cores

Figure 2 shows the example of a service "X", where the same API and the same code is called directly by the RTE on different cores. This is the default, if the services (respectively the OperationInvokedEvents) are not mapped to a task.

The code must be reentrant and concurrency safe, which means that all access to data must be protected against concurrent access by the same or by a different entity of the same module.

In this example, the same service "X" (BswCalledEntity) writes into a module internal data buffer accessible from core 0 and from core 1. A "main function" (BswSchedulableEntity), which is mapped to a task, reads the data from the buffer for further processing. In order to prevent read/write-conflicts, this “main function” must be protected from reading the buffer while it is written.

This can be considered a special case of the generic master/satellite approach for systems without memory protection between the BSW partitions.

The advantage of this approach is that the original, unchanged modules can be used, as long as they are implemented concurrency safe, which is usually the case for single core already, if different entities of the same module work on the same data, as shown in the example for core 0. Compared to the AUTOSAR R4.0 solution, where all service calls have to be routed to the master core, the performance can be improved considerably without much effort (assuming there is no need to do cross-core communication later).

The following must be considered for a concurrency safe, reentrant implementation:

- Access to all shared resources, e.g. buffers, is protected by ExclusiveAreas.

- Call trees can be made multi-core safe, if either called entities are safe, or calls are protected by ExclusiveAreas (if lock times stay within a specified limit).

BswCalledEntities that are available to CDDs can also be called directly by the CDD. The same rules apply as in R4.0.

The SchM must support cross core ExclusiveAreas, implemented by protected Spinlocks. A protected spinlock is an exclusive area that has "OS_SPINLOCK" as its value of "RteExclusiveAreaImplMechanism". This kind of exclusive areas is available for controlled access by BSW modules only. Protected spinlocks are handled by the Basic Software Scheduler.

3.5 Accessing Hardware/Drivers

BswModuleEntities of the MCAL (drivers) are accessed within the BSW partition where the caller is located.

If the same driver is required in different BSW partitions, different types of implementations are possible:

- The same reentrant code can be executed in each BSW partition. The driver can be accessed with the same API in each BSW partition. The code does not need to be partition-aware. This is the default solution, which is safe as long as the hardware objects are exclusively assigned to a single partition.
- The driver is of master/satellite type with internal branching, depending on the BSW partition it is running in. The masters may call other entities in the same BSW partition and access the hardware. The satellites may forward the request to the master in a different BSW partition without accessing any hardware directly.

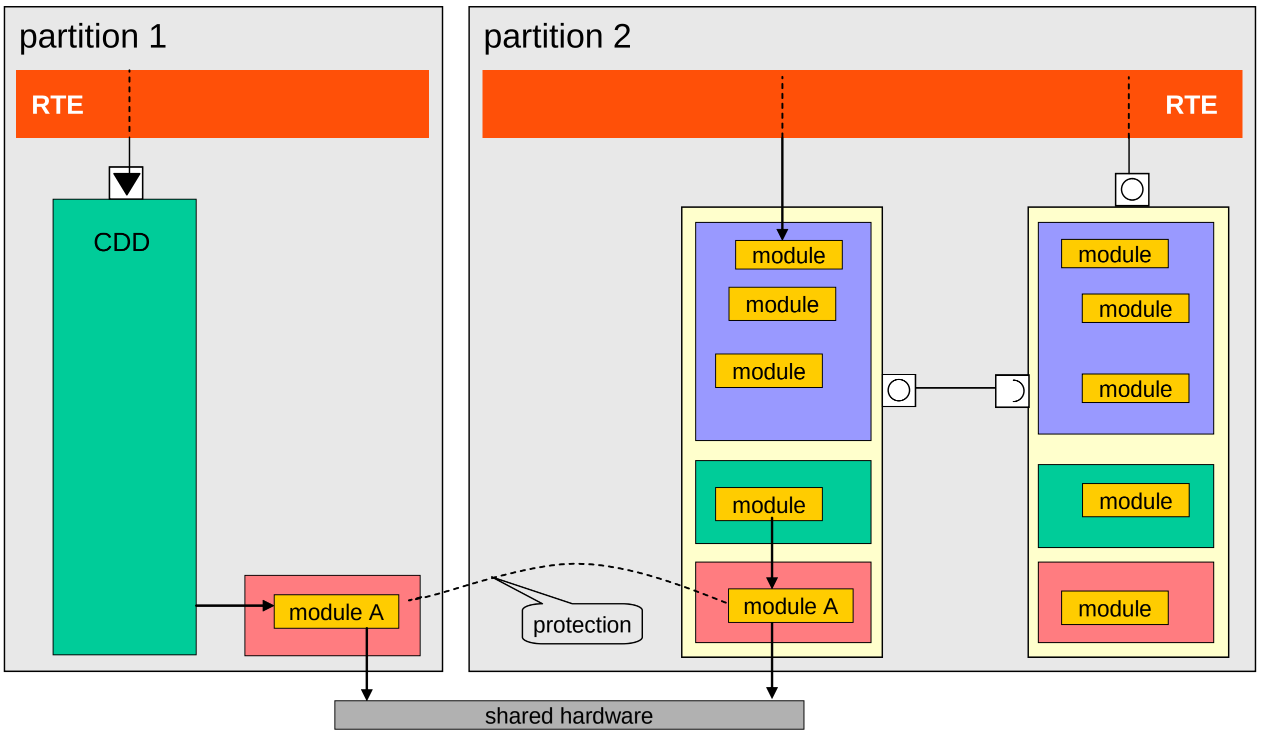

Figure 3: Protected Access to Shared Hardware

In general, the same hardware should only be accessed from one BSW partition. However, if concurrent access to the same hardware from different partitions is unavoidable, this needs to be protected within the MCAL module as shown in Figure 3, e.g. by using ExclusiveAreas. This is particularly important if drivers on different cores access the same hardware.

3.6 Concurrency safe implementation of modules

Concurrency safety of BSW modules respectively the functions implemented by these modules may be achieved by different mechanisms.

Generally, the following levels of reentrancy can be distinguished according to (TPS_BSWMDT_04103). The concrete level of a BswModuleEntity is defined in the optional attribute “reentrancyLevel”.

- **Multi-core reentrant:** Unlimited concurrent execution of an interface is possible, including preemption and parallel execution on multi-core systems. This level can be either achieved by mutual exclusion when entering critical regions, or by the absence of such regions, for example if there are no shared resources (including hardware and memory).
- **Single-core reentrant:** Pseudo-concurrent execution (i.e. preemption) of an interface is possible on single core systems. This is the highest level of reentrancy defined by AUTOSAR 4.0.3. Because it does not explicitly cover multi-core systems, “concurrency safe” has been introduced additionally. This level can generally be ensured by the same mechanisms as “concurrency safe”, but they must be ensured to work across core boundaries.
- **Non-reentrant:** Concurrent execution of this interface is not possible.

If a module that is not concurrency safe is invoked in different partitions, there is no warranty that the module will uphold its desired behavior. In this case, correct behavior shall be ensured by the usage of the module, for example if the caller(s) prevent parallel execution by using exclusive areas.

4 SchM Interfaces for Parallel BSW execution

This chapter describes the extensions to the SchM required by the concept “Enhanced BSW allocation”.

The Basic Software Scheduler (SchM) is responsible for handling the inter-partition communication between BSW modules. This is conceptually similar to the handling of inter-partition communication between SW-Cs by the RTE. Because the BSW modules are arranged below the RTE in the AUTOSAR architecture however, the communication must be available before the RTE is available. Therefore and for reasons of performance, BSW modules use the SchM for communication.

For the distribution of BSW modules across several partitions, the SchM shall implement the methods `SchM_Call`, `SchM_Result`, `SchM_Send` and `SchM_Receive`, which are used to handle service calls and callbacks as well as writing data to and reading data from a sender-receiver connection. For details on the signatures of these functions, please refer to Section 3.3, which describes the SchM extensions from a BSW developer's point of view.

The SchM can use `IocSend` (a direct call to the OS) to send data in inter-partition communication. Other RTE internal mechanism might not be available during startup.

The Inter-OS-Application Communicator (IOC) shall be configured to provide `IocSend_<Id>` functions with a uniquely determined `<Id>` for all client-server and sender-receiver connections that cross partition boundaries.

Analogously, the SchM shall use `IocReceive` to receive data from inter-partition communication, and the IOC shall provide the corresponding `IocReceive_<Id>` functions.

The following frame contains some pseudo code snippets that show how to use the IOC for inter-partition communication.

```
void some_BSW_function(){
  char *str = "some text";
  SchM_Send_Data_Src_DstN(str);
}

Std_ReturnType SchM_Send_Data_Src_DstN(char *str){
  IocSend_1(str, 5);
  ActivateTask(TASK1);
}

Std_ReturnType SchM_Receive_Data_Src_DstN(char *str){
  IocReceive_1(str);
}

TASK(TASK1){
  char data[20];
  SchM_Receive_Data_Master_Sat1(data);

  /* do something with data */
}
```

5 Configuration of Basic Software in Partitioned Systems

This is the chapter for integrators.

5.1 Task Mapping

The parallelization of BSW modules introduces several new subclasses of BswEvent to the AUTOSAR metamodel. These classes are shown in Figure 4. Each BswEvent (including instances of subclasses of BswEvent) is assigned to a BswSchedulableEntity, which is started upon occurrence of the event.

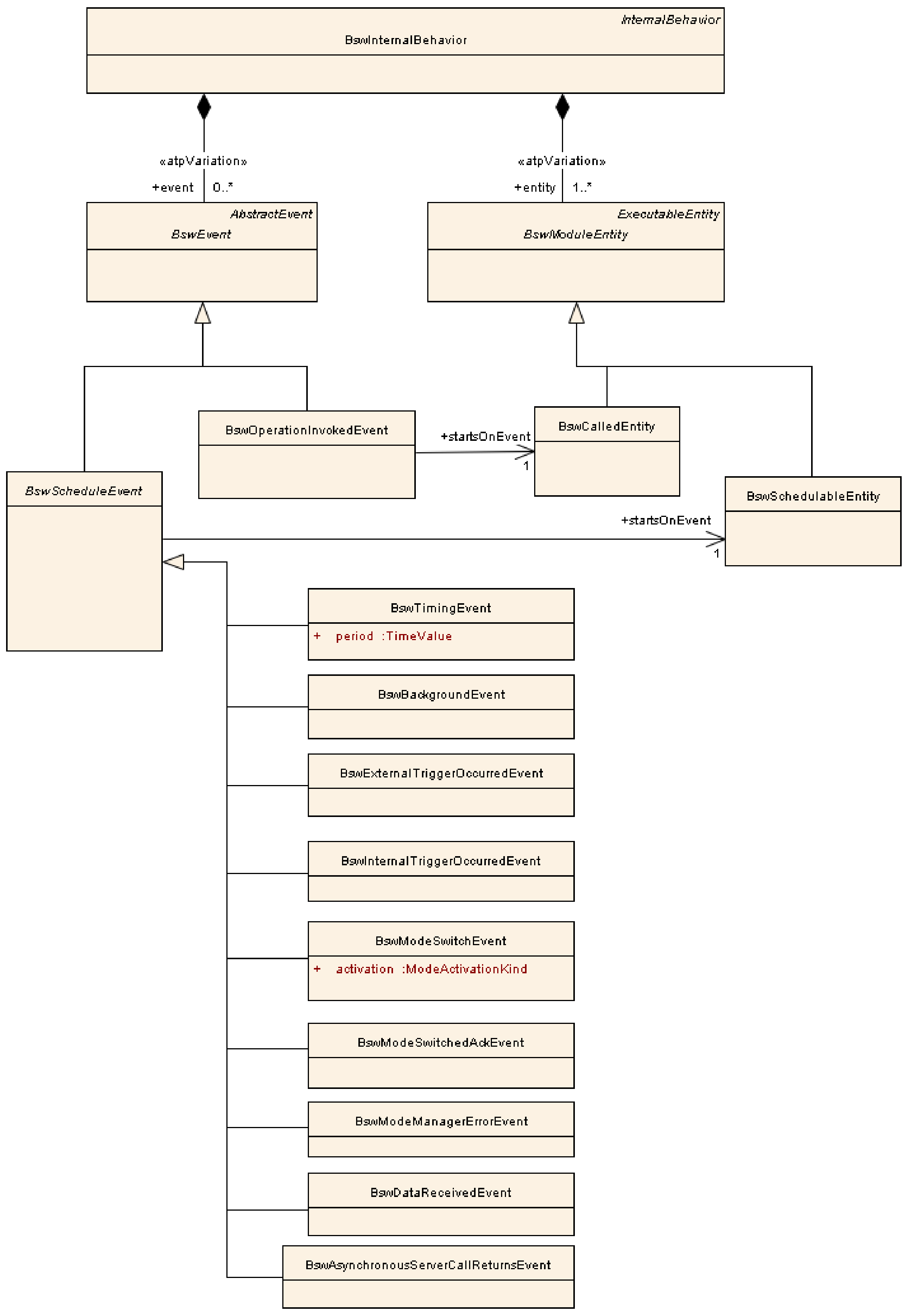

Figure 4: Events triggered by the invocation of BSW functions

A more fine grained description of the partition specific behavior of an entity can be described by the use of BswDistinguishedPartitions, as shown in **Figure 5**. A BswDistinguishedPartition is the abstract representation of a partition, which allows to the mapping of a specific BswEvent, BswModuleCallPoint or BswVariableAccess to a set of abstract partitions. The representation of a partition at this point is an abstract one in the sense that it is part of the BSW module description (according to the module description template), whereas a concrete partition is determined at ECU configuration time.

For example, if a module entity running in partition 1 provides data via a VariableDataPrototype to the same entity running in partitions 2 and 3, the BswModuleEntity aggregates a dataSendPoint with a contextLimitiation to partition 1 and a dataSendPoint with a contextLimitation to partitions 2 and 3.

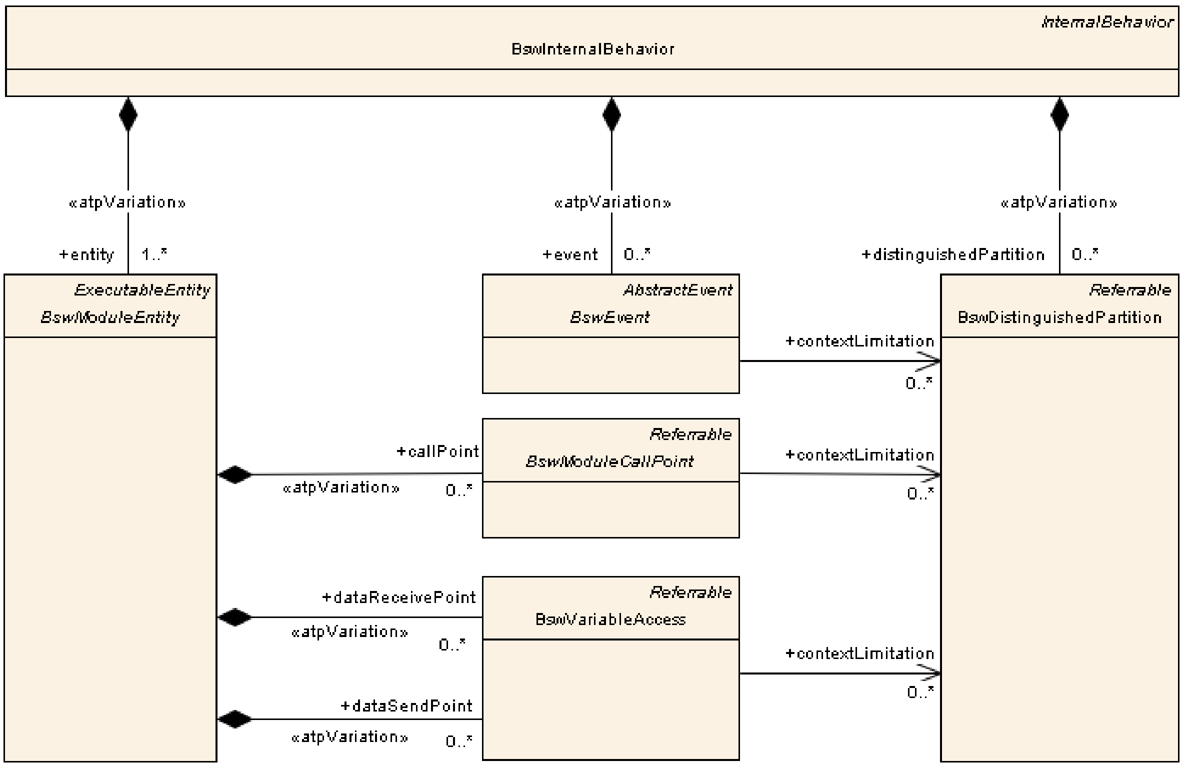

Figure 5: Modeling partition specific properties of entities using BswDistinguishedPartitions

The actual partition for the handling of an event is determined by its task mapping.

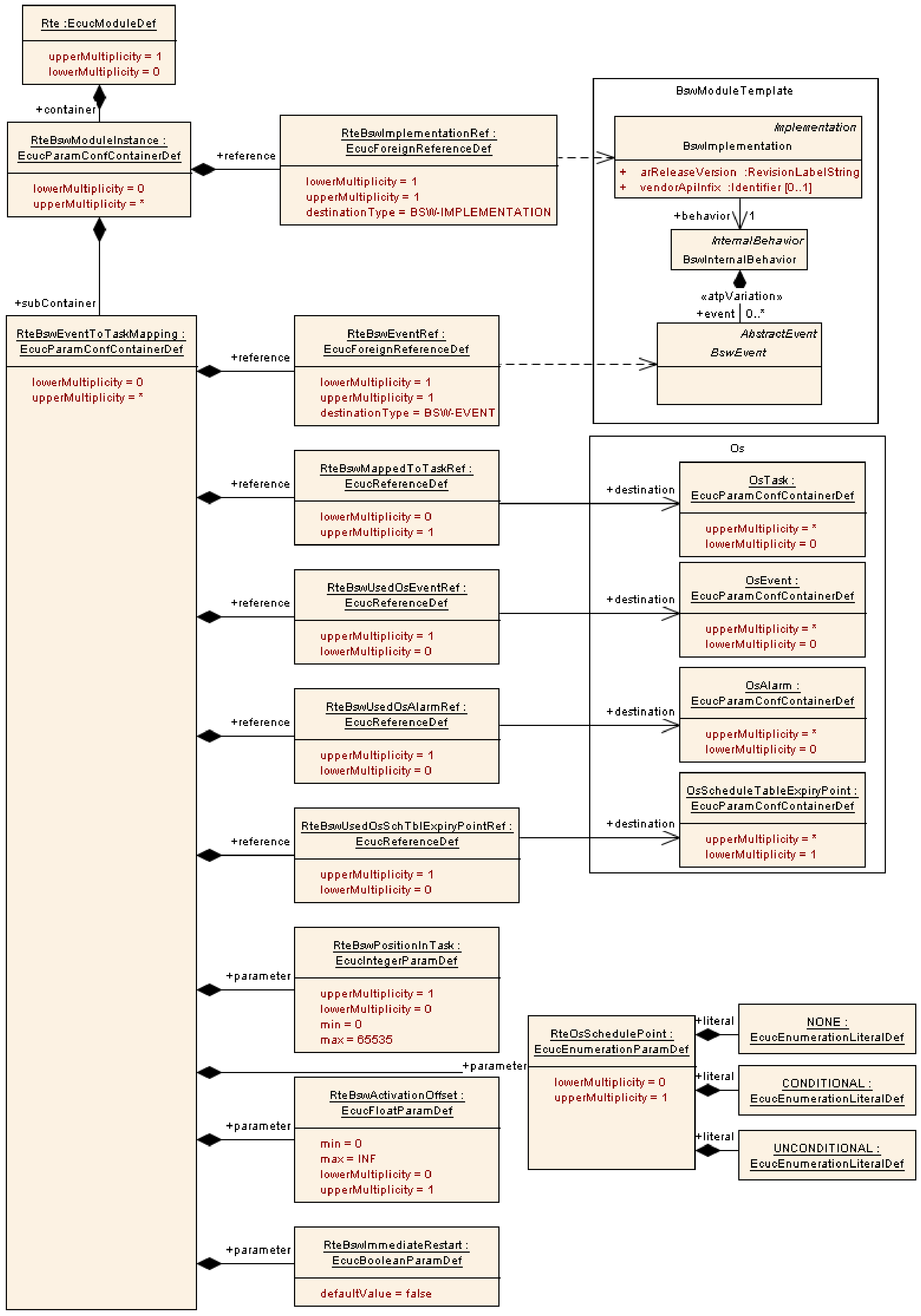

Figure 6 shows the corresponding excerpt from the AUTOSAR metamodel.

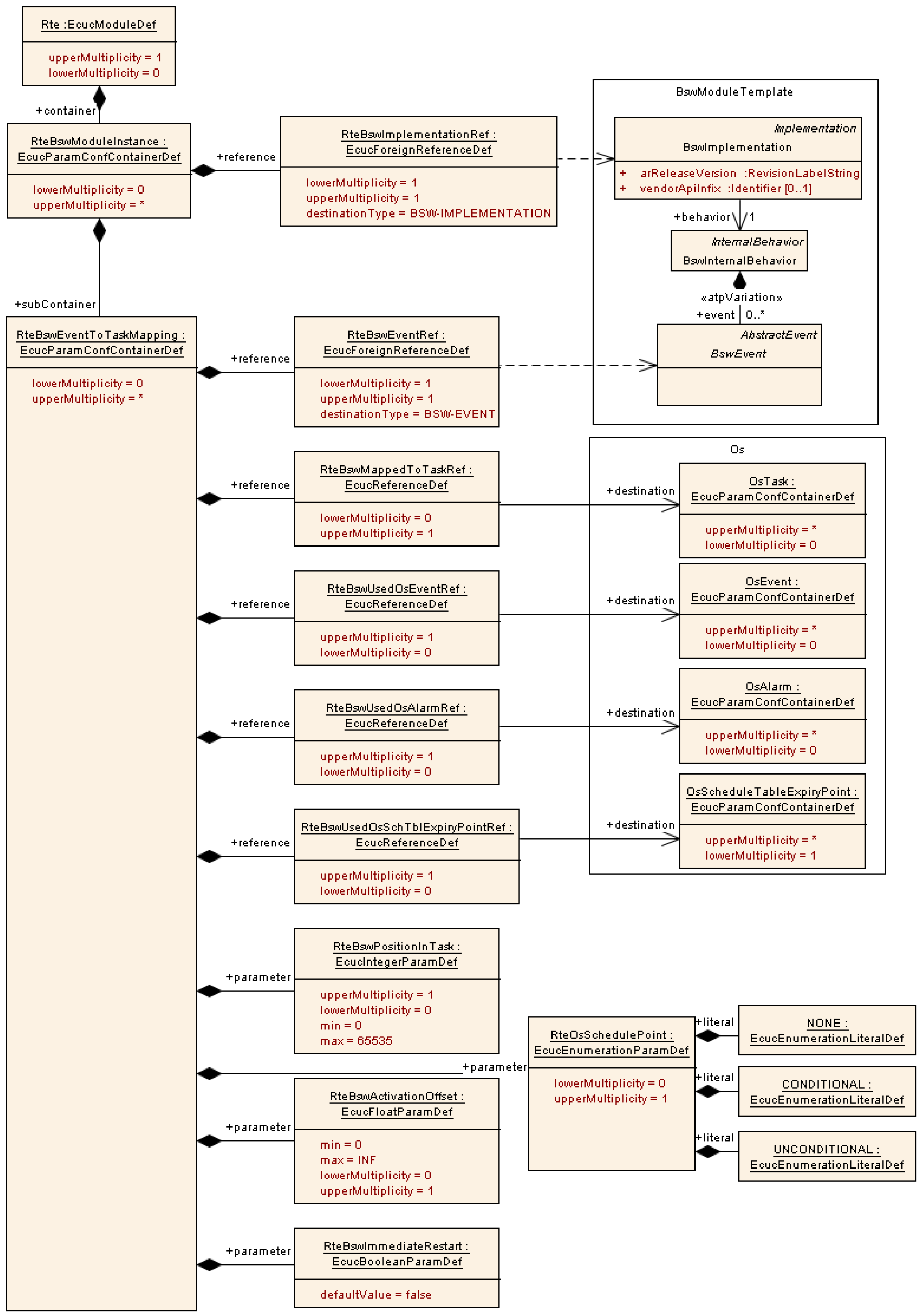

Figure 6: Mapping OperationInvokedEvents to tasks

An RteBswEventToTaskMapping refers to a BswEvent (indirectly via its RteBswEventRef) and to an OsTask (also indirectly via its RteBswMappedToTaskRef). The task is in turn mapped to a partition, and the partition is mapped to a µC core, which is the core responsible for the processing of the event. Mapping an event to a task is optional; if an event is not mapped to a task, it is handled in its originating partition. If no special mechanisms apply that prevent concurrent execution, a prerequisite for a non-mandatory mapping of an event to a task is:

- if the BSW entity is shared between multiple BSW partitions the entity needs to be *concurrency safe*
- in case it is exclusively available only on one BSW partition it needs to be at least *reentrant*.

Please note that it is currently not allowed to map RunnableEntities of a SW component to multiple partitions [SWS_Rte_07347]. For BSW it is possible to map the same module entities to different tasks and partitions by using different BSWEvents referring to the same entity

5.2 General Configuration of Master and Satellites

Modules that shall be available in multiple partitions can be implemented as masters and satellites. In this case, the master and all satellites of the same module share the same code (which may implement core-dependent behavior however) and the same configuration. Hence, a master and its satellites are treated as one module entity w.r.t. their configuration.

The communication between master and satellites is not to be standardized. It is considered to be module-internal and it is not visible to other modules. However, since it is recommended to use SchM mechanisms for internal communication, the non-standardized client-server entries and data accesses in the BSWMD to connect master and satellite need to be configured.

5.3 Configuring the BswM (per Partition)

On systems with distributed BSW there is one BSW Mode Manager (BswM) per partition (but one OS and EcuM per core, which is the same as long as we have one BSW partition per core). Each of these BswMs can be configured independently. A BswM mainly interacts with the state managers (ECU state manager and bus state managers, for instance) on the same partition.

The BswM is also responsible for the initialization and shutdown of BSW modules running in the same partition. Therefore, its configuration depends on the mapping of BSW modules to partitions.

The configuration of the BswMs is split across the container BswMGeneral, which contains shared configuration parameters of all BswM entities and BswMConfig containers, where one BswMConfig is defined for each BswM entity. Consequenty, the mapping of a BswM to its partition is defined in the corresponding BswMConfig container, which has a BswMPartitionRef pointing to the respective partition. This mapping of BswM configurations to partitions ensures that for every partition the correct configuration of the BswM can be determined.

Additional extensions to the BswM configurations for the allocation of BSW modules to multiple partitions are

- A reference BswMRequestRemoteMode in the container BswMAvailableActions. This action indicates a call to a BswM in a different partition, which is used to propagate mode requests.
- References BswMBswMModeRequest and BswMBswMModeSwitchNotification in the container BswMModeRequestSource. The BswMBswMModeRequest indicates that the source of a mode request is a BswM running in a different partition (ECUC_BswM_00980, cf. [5]). BswMBswMModeSwitchNotification indicates that another BswM has switched a mode.
- All functions listed in an action list that is processed by a BswM entity must be available in the partition this BswM is running in.

5.4 Configuring the EcuM (per Core)

On systems with distributed BSW there is one EcuM per core (even if there are multiple BSW partitions on that core). In other words, on every core there shall be one and only one partition that runs the EcuM. The partition running the EcuM is determined by the EcuMFlexEcucPartitionRef, which is specified in the container EcuMFlexUserConfig of the EcuM configuration.

Distributing the BSW is only possible when using the EcuM Flex; the EcuM Fixed does not support this.

On architectures with a sequential start of cores, there is one designated master core in which the boot loader starts the master EcuM via EcuM_init. The EcuM in the master core starts some drivers, determines the Post Build configuration and starts all remaining cores with all their satellite EcuMs.

On architectures where all cores are started at the same time, core dependent branching within the EcuM_init function can be used to achieve core-specific behavior. This can in turn be used to identify the EcuM master (running on the master core), which is responsible for the EcuM initialization on the slaves.

6 Outlook on Upcoming AUTOSAR Versions

In this chapter, we list changes to the distribution of BSW that are likely to occur in the next backward incompatible release of AUTOSAR. Hence, the content of this chapter is not applicable to AUTOSAR 4.1.1 implementations, but is supposed to show possible extensions and enhancements for future versions of AUTOSAR in that respect. Note that all these topics need to be considered in parallel, because definitions of BSW functional clusters and their standardized interfaces, which will be named "Standardized AUTOSAR BSW Cluster Interface" then, are needed to support a safety use case.

6.1 Known limitations

The support for Basic Software Allocation in AUTOSAR is currently limited to backward compatible changes (w.r.t. AUTOSAR 4.0.3). This, as well as the fact that it currently focusses on performance use cases, does currently result in the following restrictions, which may not apply to future releases of AUTOSAR:

- There is (at most) one BSW partition per core.
- BSW modules shall only run in trusted partitions.
- Communication between master and satellites is not standardized.
- BSW functional clusters and their AUTOSAR BSW Cluster Interface are not standardized.

6.2 Safety use cases

A typical safety use case deals with the integration of independent applications, possibly with different ASILs, on the same partitioned microcontroller. These applications all access BSW services (communication, I/O, memory, watchdog, or systems services).

Depending on the distribution of BSW to trusted or non-trusted partitions and on the applied mechanisms for spatial and temporal partitioning, upcoming versions of AUTOSAR will support both performance and safety use cases.

Each set of applications and BSW services could be located in separate, non-trusted partitions to prevent that faults in one set of application/BSW services do interfere with the other sets of applications/BSW services or to increase robustness and availability of the system. Within such a partition, the BSW accesses exclusive hardware resources assigned to this specific partition. Running BSW in non-trusted partitions might also allow for partitions that run both application and basic software, e.g. CDDs.

6.3 Multiple BSW partitions per core

If the BSW can be distributed to multiple partitions per core in upcoming AUTOSAR releases, BSW modules that do not implement any safety-related functionality or safety mechanism could be assigned to one non-trusted OS-Application / partition (e.g. treated as QM software). This enables freedom from interference to other software of the microcontroller. In this case, different BSW modules, SW-Cs and CDDs running in separate OS-Applications / partitions (e.g. grouped due to functional needs) can comply with different safety integrity levels (e.g. ASIL D, ASIL B, QM).

6.4 Standardized BSW functional clusters

BSW functional clusters are groups of functionally coherent BSW modules. Each BSW functional cluster includes a set of BSW modules. It is possible to have several functional clusters of the same type (e.g. several I/O clusters in different partitions), each using a different set of modules (e.g. IOHWA + ADC in one partition and IOHWA + ADC + DIO in the second partition). Each functional cluster has a ”AUTOSAR BSW Cluster Interface", which is used to communicate with other functional clusters

BSW functional clusters can be allocated to different partitions, and functional clusters of the same type can be available in several partitions. Different functional clusters can be allocated to the same or to different partitions.

The same functional cluster can only exist at most once in each partition.

But this whole cluster allocation and the resulting real interfaces are not yet standardized, just the technique is proposed here. Thus:

Upcoming versions of AUTOSAR may standardize one or more of the following:

- Define which modules are assigned to which BSW functional cluster (=> “Standardized BSW functional cluster”). It is very likely that modules of the same stack (for instance I/O services, I/O hardware abstraction and I/O drivers) will be assigned to the same functional cluster.
- Standardize communication between functional clusters of different types via "Standardized AUTOSAR BSW cluster interfaces”, as shown in Figure 7.

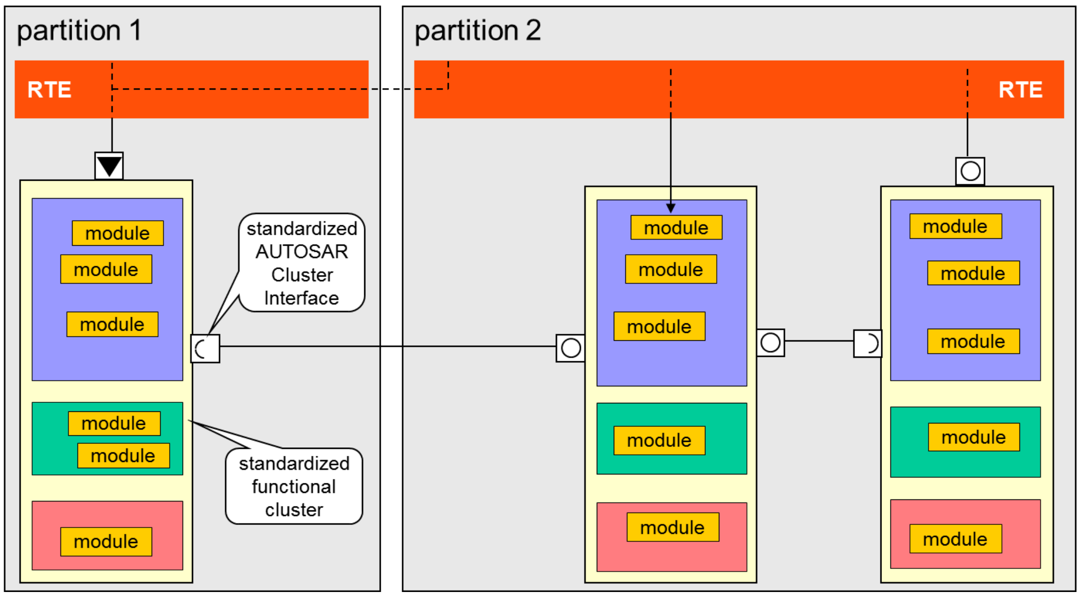
partition 1
partition 2
RTE
RTE
module
standardized AUTOSAR Cluster Interface
standardized functional cluster

Figure 7: Standardized BSW Functional Clusters

7 Glossary

All technical terms used throughout this document - except the ones listed here - can be found in the official AUTOSAR glossary [2] or the Software Component Template Specification [3].

7.1 Acronyms and abbreviations

Abbreviation	**Explanation**
ASIL	Automotive Safety Integrity Level
QM	Quality Managed (i.e. not developed according to ASIL requirements)
IOC	Inter OS-Application communicator, part of OS
MCU	microcontroller unit, µC
MCAL	microcontroller abstraction layer

7.2 Technical Terms

Term	**Explanation**
BSW functional cluster	A coherent group of BSW modules. The technique is proposed in this document, but a real allocation of modules to clusters is currently not standardized. A BSW functional cluster may be similar to what usually is called a "stack", but it would also be possible to combine several stacks into a cluster or to distribute a stack across several clusters. A BSW functional cluster includes the superset of modules, which can be part of the functional cluster, but not all modules need to be available in a specific implementation. In case the real allocation of BSW modules to BSW functional clusters is standardized in future, they probably will be named “Standardized BSW functional clusters”. BSW functional clusters can be allocated to different partitions, and clusters of the same type can be available in several partitions (either on the same or on different cores). Different functional clusters can be allocated to the same partition. Note: Contrary to ICC2 clustering, the internal structure and the interfaces between the modules within the functional cluster are not affected by the BSW multi-core support in AUTOSAR 4.1.1.
AUTOSAR BSW Cluster Interface	Interfaces between BSW functional clusters resulting from a vendor/project specific definition of BSW functional clusters. The technique is proposed in this document in a vendor/project specific way. But the allocation of modules to BSW functional clusters and thus the resulting interfaces are not standardized yet (if possible at all). This term may be defined in an upcoming release of AUTOSAR as “Standardized AUTOSAR BSW Cluster Interface” after standardization. Contrary to the standardized AUTOSAR interfaces, AUTOSAR BSW Cluster Interfaces shall not be connected to SW-Cs or BSW modules on other MCUs.
Master	Part of a distributed BSW module that coordinates requests by

	satellites and can filter or monitor incoming satellite requests. The master may work properly even if the satellites are not available. In future versions of AUTOSAR, where case partitioning may be used to enhance safety, it may be recommended or mandatory to locate the master in a partition with a high trust level, e.g. in a trusted partition.
Satellite	Part of a distributed BSW module. The distribution of work between master and satellite is implementation specific. One possibility is that the satellite only provides the interfaces to the other modules and routes all requests to the master and answers back to the other modules. In a different scenario, the satellite can provide the full functionality locally and only synchronizes its internal states with the master if necessary. Intermediate forms between these two scenarios are possible, but the satellites in general cannot work without the master.

8 References

[1] Requirements on Basic Software Module Description Template
AUTOSAR_RS_BSWModuleDescriptionTemplate

[2] Glossary
AUTOSAR_TR_Glossary

[3] Software Component Template
AUTOSAR_TPS_SoftwareComponentTemplate

[4] Concept Enhanced BSW Allocation
AUTOSAR_CONC_EnhancedBSWAllocation

[5] Specification of Basic Software Mode Manager
AUTOSAR_SWS_BSWModeManager