

DOCUMENT

Document / Deliverable Name: **Analysis and prioritization of exploitable cross-domain services**

Document / Deliverable Nr. **D 1.2**

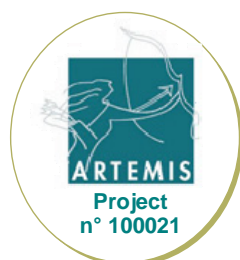
Version: draft final **1.0**

Document Type: confidential public

Responsible: **A. Balogh, OptXware**

Date of creation:: **20 October 2009**

Last modification **30 December 2009**



List of INDEXYS Beneficiaries

No	Name	Short	Country
01	TTTech Computertechnik AG	TTT	Austria
02	AUDI AG	AUDI	Germany
03	Delft University of Technology	DUT	Netherlands
04	EADS Deutschland GmbH	EADS-IW	Germany
05	NXP Semiconductors Netherlands B.V.	NXP-NL	Netherlands
06	OptXware Research and Development Ltd.	OPT	Hungary
07	Thales Rail Signalling Solutions GmbH	TRSS-AT	Austria
08	Technical University of Darmstadt	TUDA	Germany
09	Technical University of Kaiserslautern	UNIKL	Germany
10	Vienna University of Technology	TUVI	Austria

Author(s)

Name	Company
A. Balogh	OPT
R. Coelho, G. Fohler	UNIKL

Project Coordination

TTTech Computertechnik AG

Schoenbrunner Strasse 7
1040 Vienna, Austria

Technical Matters:

D.I. Andreas ECKEL, MBA
Email: andreas.eckel@tttech.com
Tel: +43 1 585 34 34 – 16
Fax: +43 1 585 34 34 – 90

Financial Matters:

D.I. Andreas BAUMGARTNER
Email: andreas.baumgartner@tttech.com
Tel: +43 1 585 34 34 – 942
Fax: +43 1 585 34 34 – 90

Copyright 2009: The INDEXYS Consortium
www.indexys.eu

Revision chart and history log

Version	Date	Reason
0.1	20/10/2009	Initial Version
0.2	24/11/2009	UNIKL input integrated
0.3	27/11/2009	Completed first version for internal review
0.4	14/12/2009	Minor alignment according to TAB recommendations
1.0	30/12/2009	Spell check, formatting, new references added

Table of Contents

1. INTRODUCTION.....	7
2. GENESYS COMPONENT ARCHITECTURE	8
2.1 GENESYS components	8
2.2 Domain model.....	8
2.3 Deployment templates	10
2.3.1 Component deployment rules	10
2.3.2 Connector instantiation rules.....	10
2.4 Non-functional properties.....	11
2.5 Definition of platform services.....	12
3. GENESYS CORE SERVICES	13
3.1 Basic Configuration Services	13
3.1.1 Basic boot service.....	13
3.2 Basic Execution Services.....	14
3.3 Basic Time Services	14
3.4 Basic Communication Services	14
3.4.1 Conceptual services.....	14
3.4.2 Candidate technologies for implementation	15
4. OPTIONAL SERVICES	17
4.1 Generic Middleware Services	17
4.1.1 Voting Service.....	17
4.1.2 Event Recognition and handling	18
4.1.3 High-level Protocol implementation.....	18
4.1.4 Receiver controlled streaming.....	18
4.2 Diagnostic Services	18
4.2.1 State externalization	19
4.2.2 Membership service.....	19
4.2.3 Analysis of diagnostic information.....	19
4.2.4 Component restart service	20
4.3 External Memory Management Service.....	20
4.3.1 Access Control of Memory Partitions	20
4.3.2 Stable Storage	21
4.3.3 Secure Storage	21
4.4 Resource Management Services.....	21
4.5 Gateway Services	21
4.5.1 Wireless connection.....	22
4.5.2 Internet connection	22
4.5.3 Legacy integration.....	22
4.5.4 Fault-tolerant clock synchronization.....	23
4.5.5 Process input/output	23
4.6 Mobility services.....	23

4.7	Security Services	23
4.8	Summary.....	24
5.	SUMMARY	26
6.	REFERENCES	27

List of Figures

Figure 1: INDEXYS domain model.....	9
Figure 2: Component instance count specification	10
Figure 3: NFP metamodel	12
Figure 4: Event recognition service architecture	18

1. Introduction

The aim of the INDEXYS project is the implementation of the principles, concepts, and methodology defined in the GENESYS project. The target domains of the project are automotive, railway, and aerospace; therefore the implementation should be focused on traditional, closed systems that are present in these areas.

In the current paper we discuss the cross-domain platform services defined by GENESYS, and analyze them from different points of view. First, we analyze the applicability of the services from the target application domains point of view, then we investigate the existing solutions and implementations in the different domains, finally we analyze the services from the modeling and development methodology point of view.

The paper is organized as follows: Chapter 2 introduces the component model of GENESYS, which is the basis of the service and application architecture and proposes a modeling approach for both service and application components. Chapter 3 describes the core services as specified by GENESYS, and Chapter 4 contains an analysis and prioritization of optional cross-domain services from the INDEXYS point of view. Finally, Chapter 5 concludes the paper.

2. GENESYS Component Architecture

GENESYS defines an architecture that features strong component orientation both at platform runtime and application levels. We introduce the most important characteristics of the GENESYS components and their interactions in the current chapter, based on the GENESYS book [Obermaisser et al. 2009]. We also introduce a *domain view*, or *domain-specific model* that encapsulates the core concept of the architecture and supports the uniform representation of service and application components and interfaces.

2.1 GENESYS components

A component in the GENESYS terminology is a self-contained entity that can be independently developed and used as a building block of a larger system. A component is a replaceable part of the system encapsulating an implementation and exposing a set of services through interfaces.

GENESYS distinguishes several types of interfaces:

- Linking Interfaces (LIFs) are used to offer and consume services between components. LIFs should have a precise specification both from the syntactical and semantic points of view. The GENESYS project suggested several methods for the syntactic description of LIFs but no established method exists for the semantic specification.
- Local Interfaces (LFs) establish connection between the component and its local environment. The specification of LFs is limited to the information that is necessary for the integration of components (e.g. the timing of the information exchanged). Modifications on the implementation of the local interfaces will not affect the specification of LIFs.
- The Technology Independent Interface (TII) is used to configure the component, e.g. to specify instance names to the component and its ports, to start, stop, or reset the component, etc. The messages that arrive to the TII communicate with the infrastructural part of the component (operating system, middleware, or hardware) but not with the application software.
- The Technology Dependent Interface (TDI) provides an interface to the internals of the component implementation. It is not relevant for the user of the component services.

From the component integration point of view, LIFs are the key interfaces of components. LF and TDI knowledge is needed for the component developer, while TII is a standard interface for the platform to control the execution of components.

2.2 Domain model

Based on the analysis of the GENESYS concepts [Del 1.1] a domain-specific model has been elaborated in order to support the description of GENESYS compliant component-based (sub)systems. The goal of this formalism is to have a compact representation of the component architecture which focuses only on the relevant concepts and relationships of these systems. The domain-specific modeling notation can be mapped to a standard modeling language (like UML MARTE as suggested by GENESYS) allowing for the interchange of models with generic modeling tools.

Figure 1 illustrates the metamodel of the component description language that is similar to the already existing languages (AutoSAR, SysML, AADL) used by the industry. The basic structure of a model is built up using *Namespaces* that help the organization of model elements and at the same time establish a hierarchical, model-level name space. *Components* represent GENESYS components (that can be either service or application level components). Components have *interfaces* that can be further specialized to LIFs, LFs, TIFs, and TDIs.

The syntax of the interfaces is described by *Ports* and their *PortInterfaces*. Ports represent message transmission or reception endpoints, while *PortInterfaces* describe the content of messages built up using *DataElements*. Each *DataElement* has a *DataType*, that is a platform-independent specification of the type.

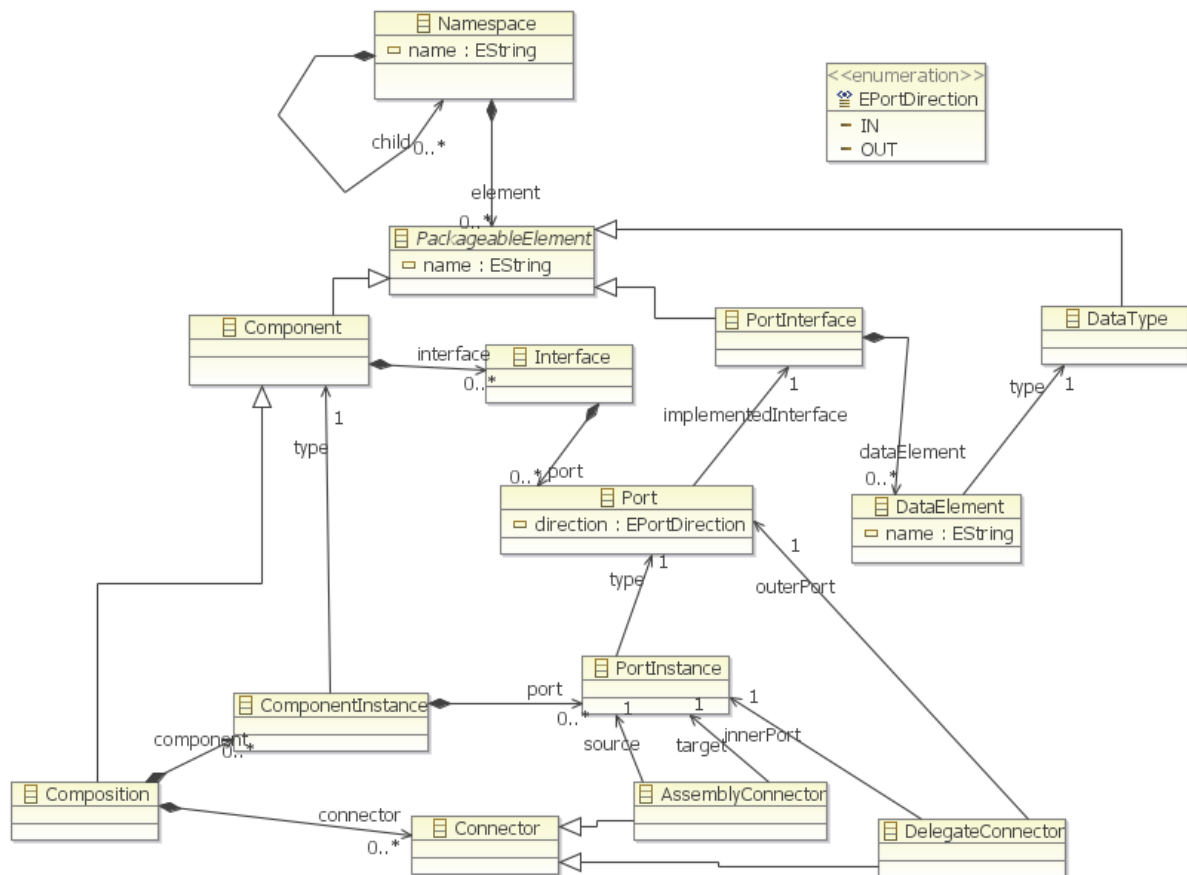


Figure 1: INDEXYS domain model

In order to allow the definition of component interaction, a new concept is introduced. *Compositions* are components that aggregate several *ComponentInstances* interconnected through *Connectors* in order to implement a complex functionality. A connector is either interconnecting two component instances (*AssemblyConnector*) or is bridging the interface of the composition and a port of an internal component instance.

The introduction of composites enables the definition and atomic reuse of complex subsystems and functionalities both on service and application levels. This structure will be used in WP1 to define the architecture of platform services and applications.

2.3 Deployment templates

The component model introduced in the previous section allows the definition of service and application components and subsystems but (as most of the existing modeling approaches) does not contain information about the instantiation of the models. For example, in case of the standard GENESYS resource management service, there are two main components: the global resource manager (GRM) and the local resource manager (LRM). On a specific instantiation of the platform, GRM should be deployed only on a single node, but LRM should be deployed to *all* execution nodes. In order to be able to include such information, an extension model is defined for the component model.

Formally, a deployment template is a component diagram (as described earlier in this paper) tagged with several *deployment rules*. These rules determine the *multiplicity* of (sub)system elements during the instantiation. We will specify several rules for components and connections in the followings.

2.3.1 Component deployment rules

Basically, a component is a self-containing functionality block of the system. It can be implemented in hardware and software, and multiple instances can be used in a single application. The key aspects that should be controlled by the deployment rules are the following ones:

- **Component instance multiplicity.** The number of instances that should be created in a specific instantiation of the component model. It can be either explicit (defining a (range of) instance number(s), or implicit (defining that at least x number of instances should be present on all computing nodes (execution nodes) of the system)
- **Component instance deployment pattern.** Specifies whether the different instances should be separated (from the dependability point of view) to different execution nodes, or they can be deployed to shared nodes.
- **Component implementation constraints.** Specifies whether the different instances should use the same implementation, or there should be at least a specific number of *different* implementations in the system. This enables the specification of homogeneous and heterogeneous replication strategies.

2.3.2 Connector instantiation rules

The components are interconnected via *connectors* representing message-based communication links. The instantiation of these links should also be controlled if multi instantiation is prescribed for the components. Several different strategies can be used for the instantiation of such connectors.

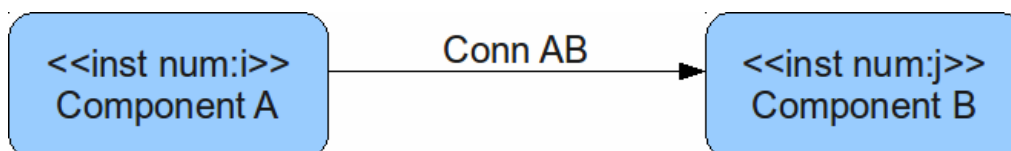


Figure 2: Component instance count specification

Figure 2 illustrates a simple situation with two components (A and B) and a connector (AB) between them. The instantiation rules of the components are represented as UML stereotypes. Component A should have i instances and component B should have j instances. The connector can have the following rules defined:

- **All-to-all.** Defines that *all instances* of the sender component should send messages to *all instances* of the receiver component. This implementation strategy is usefully for the creation of redundant configurations.
- **N-to-M.** Defines a pattern where each sender sends its messages to M receivers, and each receiver receives from N senders. A special case of this pattern is the one-to-one pattern that defines the normal point-to-point transmission pattern. It should be noted that one-to-one is only applicable if the multiplicity of sender and receiver components are equal.
- **N-to-M groups.** Several (distinct) groups should be formed each, containing N source and M target components and for each group, the all-to-all pattern should be applied.

A new problem arises with the specification of redundant or multiple links between components. If a receiver component receives multiple copies of a message the handling of copies should be defined. The following rules can be defined for the *receiver ports*:

- **Replicate ports.** The receiver port of the component is replicated in order to create a separate port for all incoming messages. This is the simplest solution from the development environment point of view, as it delegates the handling of multiple messages to the component implementation. On the other hand, it changes the *signature* or *interface* of the component that means that the existing implementations cannot be reused.
- **Voting.** Defines a voting point where the incoming message instances take part of a voting and an agreed value is relayed to the receiver. Although this mechanism introduces fault tolerance (if the sender components reside in separated execution environments) it cannot be always used. Most typically, voting is only implemented for time-triggered communication in order to avoid problems related to varying message latencies in event-triggered communication channels.
- **Multiplex queueing.** The incoming messages are multiplexed in a common queue and the receiver component receives the messages from the queue. This solution is the most suitable in case of event-triggered communication.
- **Multiplexer component.** If the receiver is a COTS component and its implementation cannot be changed, but the predefined schemes are not suitable for the receiver port implementation, a simple multiplexer component can be generated by the design environment that enables the implementation of custom solutions for message instance handling.

It should be noted that the predefined schemes can be configured to rely on existing GENESYS services (like voting) if these services are implemented in the target environments.

There is an additional constraint that should be considered during the instantiation of the component template. There are communication links that should be kept *local*, meaning that both the sender and the receiver(s) should be implemented in the same physical component. For instance, in case of an encryption service, the unencrypted data should not be sent over a network, so the corresponding connector should be marked with the **localOnly** tag.

The deployment pattern definition rules defined in the current section will be used throughout this paper to specify the structure and deployment of GENESYS services.

2.4 Non-functional properties

Several existing languages define non-functional properties (NFPs) for model elements like software components, messages, and so on that are necessary in the given application domain. These properties, however, form a fixed set that cannot be easily extended and limit the expressiveness of the models. Several attempts have been made to define common non-functional property definition formalism, but the most promising is the upcoming UML MARTE profile. The NFP package of the profile contains a generic metamodel for the definition of NFPs, including qualitative and quantitative ones (Figure 3). These properties can be instantiated and attached to any model element of the system model in order to express new aspects of non-functional characteristics.

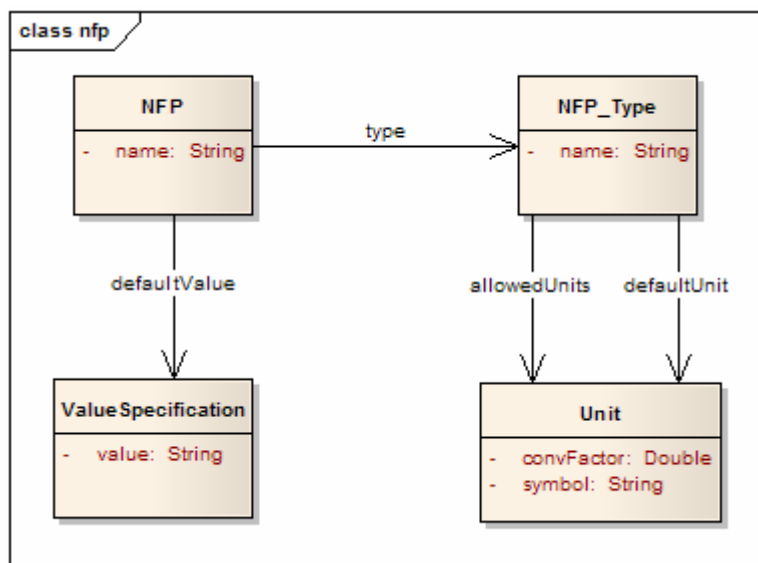


Figure 3: NFP metamodel

The MARTE NFP definition will be used as a meta model for the definition of non-functional properties in the domain models. The basic definition framework should be, however, extended with a standard taxonomy of the properties [White and Edwards, 1995] in order to have a common semantics of the models. This work is among the main focuses of WP1.

2.5 Definition of platform services

The platform services provided by the GENESYS Reference Architecture Template can be defined using two different approaches.

There are *core, essential* services that are modeled *implicitly*, meaning that they do not appear as model elements in the functional model, but are represented by other entities. For instance, the communication services are represented by inter-component connectors, the component execution and boot services as components and their properties.

Most of the services, however, follows the same component-oriented architecture and can be modeled using the same domain model as the application components. The model of such services also uses the deployment template modeling facilities in order to prescribe the instantiation constraints.

We will use both approaches in the upcoming chapters for the definition of platform services.

3. GENESYS Core Services

Core services defined by GENESYS are mandatory for all instantiations of the architecture. A so-called *trusted subsystem* should implement these services that is a compact, dependable runtime layer substituting the legacy kernels, or other operating system layers. In the current section we discuss the mandatory services from the point of view of the development methodology and tools. We suggest modeling, analysis, and synthesis methods that will be implemented during the project to provide support for the configuration, analysis, and verification of the interaction of the core services and the application components.

From the development process point of view, the core services are different from a generic platform service, as they implement functionalities that –apart from several exceptions- affect only the *runtime* behaviour of the systems, and are modeled *implicitly*. Platform services like communication, component execution, or booting are not part of the system component model, but are reflected by the combination of the functional model (communication links between components) and the non-functional description (communication and component execution timing).

3.1 Basic Configuration Services

The basic configuration services contain several essential services for the execution of applications on the GENESYS platform. Most of these services are hidden from the modeling point of view, as they will not be modeled using the component-oriented approach, but through implicit techniques.

3.1.1 Basic boot service

The basic boot service orchestrates the assignment of jobs or component instances to execution units (loading the images to the memory of a microcontroller, configuring an IP core on an MPSoC, and so on).

The allocation of functional components to execution units is either controlled by a design time configuration, or dynamically (organized by the optional resource manager services of the platform). While both cases are important, we will focus on the static, design time allocation configuration in the framework of INDEXYS, as that is relevant in all of the target domains. The design-time allocation can be created either manually, or by means of an automatic heuristics or optimization based solution.

Similarly to the boot service, the inter-component channel configurator provides also an essential functionality. The service configures the communication channels between components of the system allowing the communication between them. The communication scheme can be generated during design time (static scheduling and/or message priority assignment) or runtime (dynamic reconfiguration). As in case of allocation, the target domains of INDEXYS necessitate the usage of static communication scheme generation. The design time support for this will include various scheduling and schedule analysis tools in order to automate the synthesis and analysis of the communication subsystem.

3.2 Basic Execution Services

The basic execution services direct the lifecycle of the components (jobs) during runtime. The job *start*, *terminate*, and *reset* subservices are used to start or terminate the execution of a job, or reset it to the initial (or a known saved) state, respectively. These services are sending commands to the TII interface of the component and should be implemented by all components. These services are mainly related to scheduling, power management, time-management, and other execution control issues and can be used by higher level services like resource management or reconfiguration.

The interaction of the application components with the basic execution services will not be explicitly modeled as the services are minor, low-level elements of the GENESYS/INDEXYS architecture.

3.3 Basic Time Services

The basic time services offer basic services related to timing. *The common global time* subservice is a key component of a distributed system that maintains a unique, global notion of time within the system. This time value can be used to create timestamps that define the *temporal ordering* of events or messages. The time service can be modeled as an ordinary software component that has an output message port representing the global time source. The usage of the service can be modeled for each application component as a general message transfer.

The implementation of the common global time service can be tuned to the actual application in terms of precision and length of the timestamps. Typically, on system level (that is one of the main focuses in INDEXYS) the timestamp can be derived by the global time maintained by the core communication cluster of the system (eg. TT-Ethernet, FlexRay, TTP/C) and

The *timer interrupt service* builds on the global time service and offers temporal coordination services for the other components of the system by issuing periodic trigger events (messages) to the users of the service. Timer interrupt service usage can be modeled as part of the temporal description of the components without the need of using explicit component-oriented modeling of the service.

3.4 Basic Communication Services

Communication services provide essential functionality for all distributed systems. In this section we will introduce the GENESYS basic communication service concept, and several candidate technologies that can be used for the implementation of the concepts.

3.4.1 Conceptual services

GENESYS defined three basic communication services:

- *Periodic Exchange of Messages*, where messages are transmitted according to a predefined, periodic schedule (time-triggered approach).

- *Sporadic Exchange of Messages*, where messages are transmitted upon the occurrence of an asynchronous event (event-triggered approach).
- *Primitive Real-Time Streaming*, where information is transmitted as a continuous data flow with predefined bandwidth and jitter properties.

Even though the definitions of these classes of messages are clear, the appropriate classification of individual messages is not always straightforward. Some messages correspond to inherent strictly periodic behavior suitable directly for the TT approach. However, sporadic messages, i.e., with unknown arrival times, but minimum time intervals between consecutive instances, can be further classified either as TT or RC messages. If these messages are classified as time-triggered with a period corresponding to their worst case period, the minimum, timely correctness is guaranteed. If their actual arrival patterns at run time, however, are not strictly periodic, reserved slots will be underutilized. On the other hand, if they are classified as rate-constrained messages no time slots will be wasted but the jitter introduced on the message delivery caused by this classification might not be acceptable by the application.

The different transmission classes can be simulated based on each other. Recent research activities [Obermaisser, 2002] have shown examples of the implementation of event-triggered messaging on top of time-triggered architectures. Similarly, there are even industrial [Volcano] solutions for the simulation of time-triggered messaging on top of event-triggered protocols. The emulation of real-time streaming on event-triggered networks is part of the common internet infrastructure, as different streaming protocols for audio and video streaming exist. It should be noted, however, that GENESYS defines services that offer *quality guarantees* for all classes that is not common in the current implementations.

Work package one will investigate the conceptual services and propose possible implementation technologies that fulfill these advanced requirements. Currently the utilization of two protocols is foreseen in the framework of the project (TT-Ethernet and FlexRay) that will be investigated in the next sections.

Besides the implementation of network communication services the corresponding analysis and synthesis tools also need further investigation. Most of the protocols need complex automated tool support in order to be able to generate and/or validate the communication configuration during design time, and to verify the ongoing communication at runtime. The current work package also targets the definition and implementation such tools for the most important protocols.

3.4.2 Candidate technologies for implementation

TTEthernet [TTEthernet] is a promising technology that is able to implement all three communication services of GENESYS, as it supports

- time-triggered (TT)
- event-triggered (ET)
- and rate-constrained (RC) traffic classes.

The configuration of the protocol needs advanced scheduling algorithms. Transmission of both event-triggered and rate-constrained messages must be scheduled so that the constraints imposed by the time-triggered messages are fulfilled. The published research [Isovic et al. 1999] and [Isovic et al. 2000] will be used as start point to solve these issues. Even though these publications handle scheduling of tasks on a Central Processing Unit (CPU), the proposed methods will be investigated so that they can be applied for network scheduling.

GENESYS offers currently services for reading and writing periodic, sporadic and real-time stream messages. However, the mechanism used to schedule these messages has not been addressed yet. Therefore, new approaches for both off-line and on-line scheduling mechanisms of messages in TTEthernet are now under investigation. The application in different domains will be studied.

From the modeling point of view, the functional and architectural specification of the component model will be complemented by temporal properties for the messages that define the required temporal behavior of the communication that can be used as input for the scheduling process.

FlexRay [FlexRay] is the new standard protocol in the automotive domain, and will be used in INDEXYS WP2. FlexRay supports time-triggered and event-triggered communication modes, and the communication bandwidth can be divided between these during development time. FlexRay also needs static schedule (and thus scheduling algorithms) in order to synthesise the configuration of the communication controllers.

There are existing solutions for scheduling that need to be extended and integrated to the GENESYS compliant methodology of the INDEXYS design environment. The goal is the implementation of an automatic, optimization-based model-level scheduling tool for FlexRay that supports the definition of objective functions of different non-functional aspects (system cost, robustness, power consumption, etc.) and supports the definition of implementation or application level constraints (pre-defined schedule parts, limited number of message buffers, etc.).

4. Optional services

The GENESYS Reference Architecture Template defines several *optional cross-domain services* that can be implemented by instantiations of the platform. These services are grouped into service groups, each related to different aspects of the systems. While the services themselves are defined, there are no specific instructions on the implementation strategies or the specific interfaces of them.

In the current Chapter we enumerate these services, analyze them from the INDEXYS application domains point of view, and prioritize them. Several services are important for the current project, but there are other ones that are out of scope from our point of view. We will also give an outline of the support that WP1 will contribute to the most important services, from the modeling, analysis, and synthesis points of view.

4.1 Generic Middleware Services

GENESYS introduces the notion of *job* as the core image that is loaded onto a hardware unit in order to form a component. From the architectural point of view, the internal structure of the job is invisible, only the LIF is well-defined. However, a job is considered to be structured into several layers, including a local operating system (or simple execution environment), a middleware, and the application software. The interface between the middleware and the application code is called middleware API. A part of the middleware contains generic services that are reusable between applications and use the LIF of the component for communication.

In INDEXYS, we consider these *high level* services as service components that are used by the applications and can be *embedded* in the job of the application during system integration; therefore these services will be modeled using the component-oriented approach that is also used for application components.

4.1.1 Voting Service

The voting service supports the implementation of redundant architectures, such as the triple modular redundancy (TMR). As the chip is considered a single fault-containment region in a dependable system, this service should be implemented in higher integration levels. The TMR voting service takes the output of the three replicas of the sender job, and delivers a voted result as output for further processing.

The voting service can be used for two different purposes. Firstly, it can be used for vote-on-value, to vote on the actual value of an application-level message, secondly it can be used for vote-on-state in order to vote on the correctness of job state information.

It should be noted, that the concrete implementation strategy and algorithm (eg. average, last is the best, first is the best, etc.) that performs the voting is not defined by GENESYS, and will be considered as an input parameter for the implementation.

The service (and its derivatives for other redundancy patterns) is essential for all dependable systems and application domains; therefore we consider it as *high priority* for the current project.

4.1.2 Event Recognition and handling

This service covers the recognition and handling of different *events* in dynamic systems. The concept of the service describes that it turns *observations* made by sensors to *events* based on predefined rules and the current application context. The GENESYS specification however, does not contain any further details on the specification of rules, and the implementation of the event handling.

We suppose that the event recognition is based on the scheme illustrated by Figure 4. The sensor interface components collect the input observations, the event pattern engine tries to find a matching rule for the current context, and emits event messages to the listener components.

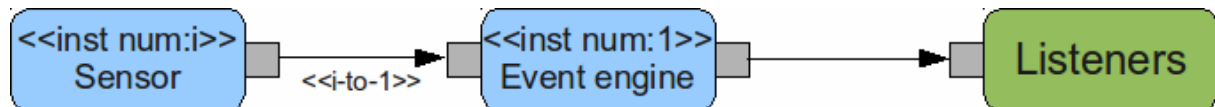


Figure 4: Event recognition service architecture

The current prototypes from the application work packages do not refer to this service explicitly, so it is considered as *medium priority* for the project.

4.1.3 High-level Protocol implementation

The core services include simple, message-based communication services. In many cases there is a need for the support of a higher level description of communication, for instance client-server (request-response) scheme, where the server awaits requests from the clients, processes them and sends response messages. In order to perform such higher level communication schemes, the configuration of basic communication channels and the orchestration of message transmissions is required. All these actions can be implemented in this service and can be transparent for the application level.

The modeling of this service can be either explicit (using the component-oriented approach), or implicit, using special ports and connectors.

As the application work packages require client-server communication capabilities, this service has *high priority*.

4.1.4 Receiver controlled streaming

The core services include the real-time streaming communication service that is extended by this service by introducing bidirectional flow control in order to handle irregular data dependant streaming traffic. Queues are employed on both ends of the transmission chain featuring high and low watermarks that are used to inform the remote party if the queue state gets critical. The queue sizes and watermarks are user configurable.

As the case studies do not involve data stream handling, this service has *low priority* from INDEXYS point of view.

4.2 Diagnostic Services

The GENESYS architecture provides several services related to *diagnostics* and *robustness* that are optional, but should be implemented in all instantiations of the architecture. The GENESYS concept

assumes that the diagnostic services are implemented in a self-contained *diagnostic component* that forms a single fault-containment region and uses the multicast messaging services to monitor (non-intrusively) the behaviour of the system.

In case of passive diagnostics, the diagnostic components monitor and record all observed anomalies to support the maintenance engineer. A fault in the diagnostic component in this case disables the event recording in the diagnostics component, but will have no effect on other parts of the system.

In case of active diagnostics, the diagnostic component can cause changes in the operation and configuration of the system; therefore a fault in the diagnostic component can be critical for the whole system.

The diagnostic services include several subservices that will be discussed in the followings.

4.2.1 State externalization

State externalization can be utilized for various purposes, including diagnosis, state exchange, global state snapshot, and so on. Components should periodically externalize their internal state at predefined recovery instants. The externalized state information can be used to 1) allow error detection and/or 2) enable checkpointing and retry mechanisms.

From the development process point of view, states are special data elements that are emitted as periodic messages from the components. The utilization of these messages depends on other services (like diagnostics) that should explicitly consume the state messages. If a replication scheme is used, these state messages can also be used to vote on the state of the replica set that involves the voting service.

As state externalization is a fundamental service for dependable systems it has *high priority* in the current project.

4.2.2 Membership service

The membership service provides a coherent global view of the operational state of the components. On chip and device levels, this information is generated by the local diagnostic component, and on system level the (fault-tolerant) membership protocol is typically part of the communication protocols (such as FlexRay, TTP/C, or TT-Ethernet).

From the modeling point of view, this service simply emits membership status messages to the recipient application components. The service is essential for dependable systems; therefore it has *high priority*.

4.2.3 Analysis of diagnostic information

The basic diagnostic services should be complemented by high-level services that analyze the messages and events logged by the diagnostic component and provide a diagnosis of the current system state. It must be ensured that all errors, even if they are masked by redundancy, are reported and logged in order to allow further analysis.

The diagnostic information analyzer should be able to interpret the input data properly, and identify transient, intermittent, and permanent errors in order to guide the maintenance and support the repair strategies.

In the framework of INDEXYS, WP1 partners will contribute the definition of (application-specific) high-level diagnosis algorithms that can be used for the implementation of this service. The priority of the work is considered to be *medium*.

4.2.4 Component restart service

If the active diagnosis is allowed, the diagnostic component can reset and restart other components in case of failures. This service defines an interface for the diagnostic component to reach the TII of the application component, and to retrieve a *restart state* for it. This service can also be used for component restart in other cases, eg. when restarting a component that has previously been switched off by the power gating strategy.

The service can be modeled as a regular component that is connected to the diagnostic component's LIF and the TII of the component to be restarted.

The priority of this service is *medium* as it is not directly required by the demonstrator application, but represents an important feature for dependable systems.

4.3 External Memory Management Service

External memories can complement in-component memories by large storage capacity. While the handling of local memory is treated by the component implementation, external memories that are independent of the component and can be shared among several application components, need proper, well-defined management.

Access to the external memories is controlled by the *external memory management* services that will be discussed below.

4.3.1 Access Control of Memory Partitions

This service partitions the external memory into separate regions with different access attributes and manages the access to these regions such that the global integrity constraints of the applications are maintained. In the basic case, when a memory region is only accessed by a single component, the access control is also simple. If, however, several different components share a common memory region, locking mechanisms should be used in order to maintain the integrity of the region. In a mixed criticality system, it should also be guaranteed that only those components having sufficient safety level can modify safety-critical data.

The implementation of this service depends on the integration level. On chip level, a dedicated hardware unit is needed to access the memories and to implement the higher level services. On device level, however, the implementation can rely on the memory management unit (MMU) of the processing units.

The memory regions should be explicitly modeled as GENESYS components and the access control parameters should be defined as non-functional properties for the interfacing components and their interactions. The configuration of the access control and partitioning component will be derived from the model automatically.

This component is essential for mixed criticality systems and should be implemented and supported with *high priority* in the current project.

4.3.2 Stable Storage

In many applications there is a need for a *stable, persistent* storage that guarantees that the persisted data remains intact even after the system undergoes several power up/power down cycles. If stable storage is provided by a component-internal element it is not considered from the architecture point of view, but in many cases it is more efficient to have a single, shared stable storage on chip or device level.

The shared storage controller is modeled as a component and communicates with the application components using messages. The stable storage can be used to periodically persist the state information of the components in order to support the quick recovery after power up or faults.

Stable storage is important in several application areas; therefore it is considered as *high priority* in the current project.

4.3.3 Secure Storage

Secure storage stores data in encrypted form in order to prevent unauthorized access to it. It can be implemented in pure software, but hardware support may also be needed in order to achieve high level of security. The security services of the GENESYS platform can be used to implement the encryption and decryption of data.

Secure storage is not considered in the demonstrator applications, so we consider it to be of *low priority*.

4.4 Resource Management Services

In addition to the basic configuration services described earlier, *resource management services* offer higher level, dynamic resource management facilities that can be used for dynamic management, reconfiguration, and optimization of the resource consumption. These services are implemented by dedicated components that have a holistic view of the system including power, energy, time, and memory, and quality-of-service management.

These services target *dynamic systems* that can be reconfigured at *runtime* in case of workload, energy, or user preference changes. The target domains of the current project, however, do not allow the inclusion of such dynamic behavior; therefore the resource management services are considered as *out-of-scope* for the current project.

4.5 Gateway Services

Gateway services cover several different features that are needed to interface a GENESYS system with other systems or with its environment. Gateways are also used to interface different integration levels *inside* a GENESYS system (eg. between chip and device levels, or device and system levels).

In general, there are several aspects of the gateway functionality that should be considered

- *Protocol translation* should be performed in order to fulfill the specification of both interfaces of the gateway
- *Address mapping* should be implemented in order to bridge between the two different addressing schemes

- Similarly, *name mapping* should also be done in order to convert between the two different namespaces
- *External clock synchronization* can also be performed in order to align the lower integration level clock to the outside worlds (e.g. GPC clock)
- *Firewall* functionality should also be implemented in order to protect the internal network from malicious intruders or unnecessary network traffic.

Several subservices will be discussed in the following subsections that implement these services.

4.5.1 Wireless connection

In open systems where the devices interact with each other or with dedicated *base stations* it is necessary to have *wireless connection* services. The wireless gateway inside the device contains the physical sender/receiver devices and should manage it to achieve a desirable error rate while minimizing the power consumption. The detection of new communication peers dynamically is also the task of this component. Wireless services are *out of scope* in INDEXYS.

4.5.2 Internet connection

A standard component that provides connection to the Internet (or any other TCP/IP based network). Its features include *protocol conversion, name and address translation, and firewalling*. This service can rely on a wireless or wired connection in order to establish a connection to the external network. Since the unidirectional message transport concept of INDEXYS can be well fit to the Internet *fat sharing* concept, the interconnection of the systems is straight forward. It should be noted, however, that additional security-related services should be employed in order to authenticate and authorize external users trying to connect to the system from the external network.

Currently, there is no direct need for Internet/Intranet connection in the planned demonstrator applications, so this service is considered as *low priority*.

4.5.3 Legacy integration

This service is used to integrate legacy and GENESYS components in a single system by interfacing the legacy networks with the GENESYS compliant chips or devices. The legacy integration gateway performs protocol conversions, and resolves all possible property mismatches between the two subsystems.

Legacy integration is a key issue in the current applications domains as this aspect must be resolved to enable the utilization of the GENESYS results in an existing application domain by integrating it to the legacy components. The possible integration schemes of this service will be investigated in the current work package, with special emphasis on the standard runtime platforms of the target application domains (like AUTOSAR in case of the automotive domain). The development environment should support the configuration of such legacy gateway components from both the GENESYS and the external interface point of view.

This service is a key enabler factor for the architecture so it is considered to be of *high priority*.

4.5.4 Fault-tolerant clock synchronization

The high availability of the global time base is a critical issue in safety-relevant systems, so it should not depend on the correct functioning of any particular clock. Synchronization theory [Lamport et al. 1982] requires that at least four independent clocks are present in order to mask Byzantine failure of any one clock (less strict solutions relying on only 2 clocks, but not covering such wide set of possible failures are also known in the industry) This implies that at least distinct FCR-s (e.g. at least four GENESYS MPSoCs) should be present and connected via redundant, reliable communication links in order to perform FT clock synchronization that will be performed on each of these nodes in order to derive the fault-tolerant global time base.

This service is modeled using the standard component-oriented approach and can be instantiated in a networked system. As it performs a key functionality for safety-relevant systems, it is considered to be *high priority*.

4.5.5 Process input/output

Process input/output components are used to interconnect external sensors and actuators to the system. These components perform device driver and interfacing functionality, convert the input/output data to the representation of the external device to that of the GENESYS system, and send (or receive) messages containing the values on their LIFs.

The process I/O components can also be used to monitor external objects and disseminate event messages in case of significant events on the observed entity. This functionality can use the basic, sporadic message transmission mechanism of the platform.

The optional hot insertion and hot replacement service can be utilized to support the addition, removal, and replacement of components for maintenance reasons without having to power down the system. Sensor/actuator interface components can also benefit from this functionality.

Sensor/actuator interfacing is a core feature, and will be supported by the design environment. The process I/O components can be modeled as regular GENESYS components having local interfaces connected to external peripheral units. The priority of this service is *high*.

4.6 Mobility services

The mobility related services represent an important aspect of *open, mobile* systems such as smart phones and sensor networks. The cross-domain services defined by GENESYS are covering the dynamic component and service detection and connectivity management related functionalities. As the current project focuses on systems employing only wired communication and closed systems, these services are *out of scope* in INDEXYS.

4.7 Security Services

Security related services group several subservices in order to support different security related functionality in GENESYS components and systems. These services are based on the tamper proof unique ID core service and should be implemented as a dedicated component. The concrete algorithms and security schemes depend on the application context.

Security is not a concern in the target domains, so these services are *out of scope* in the current project.

4.8 Summary

Several cross-domain services have been defined by GENESYS that are divided into *core* and *optional* services. All core services should be implemented by each instantiation of the architecture, and will be supported by the design environment that will be developed in the framework of work package 1.

Category	Service Name	Priority
Diagnostic Services	State Externalization	High
	Membership Service	High
	Analysis of Diagnostic Information	Medium
	Component Restart Service	Medium
External Memory Management Service	Access Control of Memory Partitions	High
	Stable Storage	High
	Secure Storage	Low
Security Services	Secure Key Management	Out of scope
	Encryption and Decryption	Out of scope
	Random Number Generation	Out of scope
	Service Authentication	Out of scope
	Secure Boot Service	Out of scope
	Service Access Control	Out of scope
Resource Management Services	Local Resource Management	Out of scope
	Global Resource Management	Out of scope
	Device Level Resource Management	Out of scope
	Configuration and Reconfiguration	Out of scope
Gateway Services	Wireless connection	Out of scope
	Internet Connection	low
	Legacy Integration	High
	Fault-tolerant Clock Synchronization	High
	Process Input Output	high
Mobility Services	Component/Service Detection	Out of scope
	Connectivity Management	Out of scope
	Mobile Device Controlled Mobility	Out of scope
	Infrastructure Controlled Mobility:	Out of scope
Generic Middleware Services	Voting Service	high
	Event Recognition and Handling	medium
	High-Level Protocol Implementation	High
	Receiver Controlled Streaming	low

Table 1: Summary of optional GENESYS services

The *optional* services have been analyzed and prioritized in the current Chapter, and will be handled according to their priority. Table 1 summarizes the services and their priorities in the context of the current project. High priority services should be supported by the development environment in first

order, then the additional services should be added in the order of priorities. There are several services that are *out of scope* of the current project; therefore they will not be addressed.

5. Summary

The core and optional cross-domain services defined by GENESYS have been discussed in this paper. We collected all the services, and analyzed them from the INDEXYS point of view. Priorities have been attached to the optional services in order to select the most important ones for support in WP1.

A domain model for the specification of component-oriented systems has also been presented that is complemented by a framework of *non-functional properties* that describe various aspects of the systems, and a deployment template that supports the parametric definition of *reusable subsystems* on the modeling level. The proposed formalism is able to represent both applications and platform services enabling the seamless migration of reusable *design patterns* to the available *service library*. The analysis of the platform services resulted in the selection of the most important optional services (related to dependability, legacy reuse, and communication) that should be supported by work package one by modeling, analysis, and synthesis methods and tools. The results of this work will be used in the next phases of the project.

6. References

[Clark 1988]	David D. Clark, The Design Philosophy of the DARPA Internet Protocols, Computer Communications Review 18:4, August 1988, pp. 106–114
[Obermaisser et al. 2009]	Roman Obermaisser, Hermann Kopetz (eds.), GENESYS: A Candidate for an ARTMEIS Cross-Domain Reference Architecture for Embedded Systems (in print)
[Isovic et al. 1999]	D. Isovich and G. Fohler, Online Handling of Hard Aperiodic Tasks in Time Triggered Systems, In Proceedings of the 11th Euromicro Conference on Real-Time Systems, 1999.
[Isovic et al. 2000]	D. Isovich and G. Fohler, Efficient Scheduling of Sporadic, Aperiodic and Periodic Tasks with Complex Constraints, In Proceedings of the 21st IEEE RTSS, 2000.
[White and Edwards 1995]	Stephanie White and Michael Edwards. A requirements taxonomy for specifying complex systems. Engineering of Complex Computer Systems, IEEE International Conference on, 0:373, 1995.
[Del 1.1]	A. Balogh, Gy. Csértán, A. Ökrös, Z. Balogh, P. Bokor, G. Fohler, R. Coelho. Report on initial project alignment according to final GENESYS results. INDEXYS Deliverable 1.1
[Lamport et al. 1982]	L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. ACM Transactions on Programming Languages and Systems , 4(3):382–401, July 1982.
[Obermaisser 2002]	R. Obermaisser, CAN emulation in a time-triggered environment. In "Proceedings of the IEEE International Symposium on Industrial Electronics (ISIE'02)", Volume 1, pp. 270-275. L'Aquila, Italy. July 2002.
[Volcano]	Mentor Graphics. Volcano Target Package (VTP) for CAN & LIN http://www.mentor.com/solutions/automotive/tools/networking
[FlexRay]	FlexRay Consortium. FlexRay Protocol Specification V2.1 Rev A. 2005.
[TTEthernet]	TTTech Computertechnik AG. TTEthernet Specification. 2008.