A Comparison of NoTA and GENESYS

B. Huber
Vienna University of Technology, Austria
{huberb}@vmars.tuwien.ac.at

Abstract

The trend of converging application domains in the field of embedded systems has been observable during the last years. For instance multimedia systems have entered the automotive domain and interfaces to the Internet are included in many embedded control applications. In the last years, undertakings to tackle the challenges introduced by this digital convergence have been started by the European Union. The GENESYS (Generic Embedded Systems) project is an European project that aims at providing a basis to solve these challenges by developing a cross-domain architecture for embedded systems. This paper compares the cross-domain GENESYS architecture with the ambitious Network on Terminal Architecture (NoTA), which has been primarily developed for the mobile consumer electronics domain. The paper elaborates on the commonalities and differences of both architectures and discusses the possibility of combining both approaches.

1. Introduction

Over the past years a enormous increase of the significance of embedded systems has been identifiable. Mainly due to the tremendous increase of the performance, the shrinking geometries and the decreasing costs of embedded computing devices, many in recent history unimaginable functionalities are realized by embedded systems nowadays. This trend is observable across application domains: The consumer electronics domain has experienced an enormous boost due to the ubiquitous mobile applications. Control applications benefit from the increasing performance by the ability to solve more complex control tasks and from the decreasing costs by the ability to increase system dependability by deploying replicated nodes with lower cost overhead.

This advancements in embedded systems have led to specific experts and technologies in the individual application domains. However, nowadays a opposite trend is observable: For instance multimedia systems are integrated in automotive systems and not only in high-end luxury cars (e.g., playback of audio stream from dynamically joined audio sources like MP3 players, integration of mobile telephony in the entertainment system of the car, etc). On the other hand, control applications make use of Internet technologies, for instance to enable the remote observation, diagnosis and maintenance of industrial plants.

This trend imposes new challenges on the entire embedded systems industry. For this purpose, European industry in conjunction with the European Commission has set up the Technology Platform ARTEMIS, which defines in its Strategic Research Agenda [1] medium-term visions and research targets for the European embedded system industry, including the development of a cross-domain reference architecture for embedded systems. As a first initiative to tackle the high priority research challenges defined by ARTEMIS [2], the European project GENESYS[1] has been started in January 2008. The objective of GENESYS is to develop a blueprint of such a cross-domain architecture.

Independent of the ambitious efforts of ARTEMIS, the development of an architecture, mainly tailored to the needs of the mobile consumer electronics domain, has been started in 2003. This Network on Terminal Architecture (NoTA)[2] addresses very similar objectives, including the increase of interoperability, composability and reuse of systems in order to shorten the time-to-market and reduce development cost as well as to provide a platform for application development that is nearly agnostic to changing implementation technologies. This paper presents a comparison of both architectures – GENESYS and NoTA – and elaborates on potential synergies.

The rest of the paper is structured as follows. Section 2 introduces Network on Terminal Architecture (NoTA), which is a modular, service-based architecture

1 `http://www.genesys-platform.eu/`
2 `http://www.notaworld.org`

for embedded systems. Section 3 is devoted to Generic Embedded Systems (GENESYS), a European cross-domain architecture. The commonalities and differences of both architectures are discussed in Section 4. The feasibility of combining NoTA and GENESYS is addressed in Section 5. The paper finishes with a conclusion in Section 6.

2. Network on Terminal Architecture

In the consumer electronic domain, the Network on Terminal Architecture (NoTA) is a modular service based system architecture for mobile and embedded devices. It is an open architecture with the primary goal to define a unified interface for embedded devices in order to ease the development and integration of interoperable services and devices. The development of NoTA is driven by an open architecture initiative, initially started at Nokia Research Center in 2003, with the aim to provide a solution that can be used throughout the industry, academia and developer community.

NoTA does not define services for any specific domain or products, but provides a service-oriented framework for the design of embedded application, which is driven by end-user requirements [3]. NoTA identifies devices that consist of service nodes and application nodes. Devices communicate via the so-called Device Interconnect Protocol (DIP). The DIP offers two communication modes: message-based communication and streaming communication. The message-based communication is bi-directional and used by application nodes to exert control over service nodes. The streaming communication is uni-directional and enables the transfer of data (e.g., multimedia data).

2.1. System Structuring

For system design NoTA provides three distinct abstraction levels, denoted as *functional architecture*, *logical architecture*, and *implementation architecture* [4]. The functional architecture describes the functional aspects of the system by means of application nodes (AN) and services nodes (SN) interconnected by the DIP. Application nodes interact with the user of the system and make use of the services provided by the SNs. Service nodes provide their services to ANs and SNs and can utilize the services of other SNs to fulfill their specified service. Services provided by ANs and SNs are solely exploited via service interfaces. These interfaces are described by the Service Interface Specification (SIS), which defines the service syntax, the behavior (time-free via finite state machines) and gives

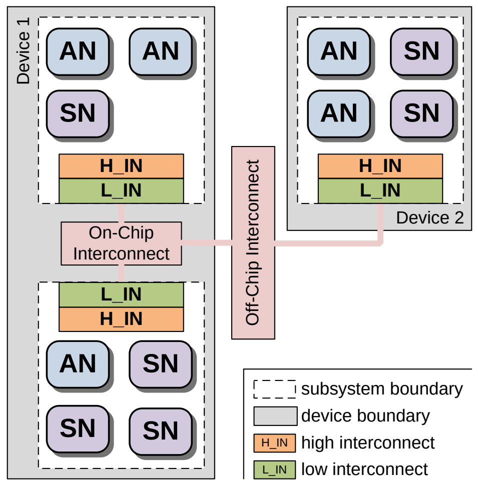

Figure 1. NoTA (Physical) System Structure

bounds for non functional properties such as the latencies, the bandwidth and the energy consumption [5].

In the logical architecture a grouping of ANs and SNs to *subsystems* is performed. Besides this logical view point, subsystems also exhibit a physical viewpoint: All resources required for implementing the ANs and SNs are part of the hosting subsystem, which are not shared with other subsystems. Hence, services located in different subsystems can access shared resources (e.g., memory) only via the service interface. Thus, the partitioning of services into different subsystems is a first important design decision influencing system performance versus service independency and encapsulation.

The implementation architecture describes the physical implementation of the system. At this abstraction level, the conceptual element provided by NoTA is a *device*. A device is a physical entity that provides resources like processors, buses, memories, and peripherals, which can host one or more subsystems. An exemplary implementation architecture of a NoTA system is depicted in Figure 1.

2.2. Device Interconnect Protocol

The Device Interconnect Protocol (DIP) represents the communication infrastructure of a NoTA system. The DIP consists of two protocol layers – the Low Interconnect (L_IN) and the High Interconnect (H_IN) (cf. Figure 1). Essentially, the L_IN serves for connecting subsystems by mapping the communication requests of services to the actual physical communication infrastructure. For this purpose it provides uniform socket-based communication mechanisms via the *Low Interface (L_IF)* to the H_IN. In order to establish communication between services, the L_IN is responsible for the discovery of the physical entities that are the endpoints for the communication activities, the so-called *Interconnect Address*. For providing a uniform interface to the H_IN independent from the underlying physical transport protocol, the L_IN is internally split in two layers. While the higher layer provides those services of the L_IN that are independent from the transport protocol and remains stable, the lower layer is tailored to the characteristics of a particular physical interface. Hence it needs to be updated whenever the physical transport protocol is changed.

The main purpose of the H_IN is the registration, discovery, and activation/deactivation of services. Analog to the L_IF of the L_IN the H_IN provides the *High Interface H_IN)* to application and service nodes. This interface consists of ports, which offer streaming and message-based sockets to the services. Application and service nodes can use multiple ports for data dissemination and/or reception. Service registration and discovery is managed by one dedicated H_IN, which is denoted *HManager*. A new service is registered at the HManager with its service ID together with its interconnect address, i.e. the information on which device and subsystem the particular service is located. For service discovery, a query containing the needed service ID is sent to the HManager, which resolves the according interconnect address. Service IDs are allocated based on service ontologies by a dedicated service node, the *Service Level Resource Manager (RMSN)* [6].

Conceptually, the DIP is mostly independent of a specific communication technology. Hence, it is possible to replace for the implementation of the DIP an off-chip network with an on-chip network with low overhead, e.g., when a chip can be replaced by an IP core due to improvements in the semiconductor industry. In such a case, the transport protocol specific part of the L_IN needs to be adapted, whereas the remaining parts of the DIP, in particular the interface towards the application and service nodes, remain unaffected.

3. Generic Embedded Systems Architecture

GENESYS (GENeric Embedded SYStem Platform) is a FP7-ICT collaborative project that has started on January 1st 2008. Its aim is the development of novel technologies for embedded computer systems that are of universal importance across individual application domains. The objectives of the project are driven by the aim to enable significant improvements concerning time-to-market, cost, and robustness for a wide range of applications, from mobile phones to safety-critical computers in airplanes. The ultimate goal of the project is to develop a cross-domain reference architecture for embedded systems that can be instantiated for different application domains to meet the requirements and constraints documented in the ARTEMIS strategic research agenda. This ambitious goal is supported by 23 project partners from all over Europe including major European companies and leading universities and research institutes.

In the following, some important architectural principles that characterize the GENESYS cross-domain architecture are introduced [7].

3.1. Component-Oriented Architecture

From the hardware and software point of view, the GENESYS architecture follows a strict component orientation. In GENESYS, a component is a *self-contained subsystem* that is used as a building block in the design of a larger system. It encapsulates the implementation and hides the implementation details. The provided services are exposed via a set of interfaces. Since a component hides the (complex) internal structure from its user, it offers an appropriate unit of abstraction for system design. Therefore, it helps to tackle the challenge of design complexity in modern embedded systems.

In GENESYS hard and soft components are distinguished. Hard components are hardware entities that cannot be modified after design and deployment. The functionality of soft components, on the other hand, is determined by software – either on an FPGA or a CPU. Hence, the functionality of soft components can be modified during the lifetime of the component. Whereas hard components may be superior with respect to performance, power efficiency, or security, soft components are important for dynamic reconfiguration in order to allow systems to evolve and adapt to a changing context.

3.1.1. Message-based Communication Infrastructure The communication infrastructure takes a key role in the GENESYS architecture, since it enables the integration of the individual components to the overall system. For reducing system design complexity, GENESYS strives for a consequent decoupling of the (computational) components from the communication infrastructure in order to facilitate the design and analysis of both parts in isolation. This decoupling is realized in GENESYS by using message passing, i.e., the unidirectional exchange of messages, as the primary communication paradigm. The exchange of messages is determined by the interface design of the components; thus, the communication infrastructure is unaffected by modifications of the internals of components as long as the behavior at the interface remains unchanged.

The message paradigm provides a higher level of abstraction compared to bus interfaces. Every message has a specific identifiable sender and one or more receivers. Multicasting is in particularly required in many real-time applications, e.g., for fault-tolerance by active redundancy or for the non-intrusive observation of component interactions by an independent observer for diagnostic purposes. The GENESYS architecture supports three distinct communication modes: *time-triggered messages* where the instants of transmission are determined by an a priori defined, collision-free message schedule, *event-triggered messages* where the transmission of messages can be triggered by any significant event, and *synchronized data streams*, which enable the on-the-fly processing of synchronized streaming data from multiple sources. The message paradigm is a universal model, since on top of a basic message passing service it is possible to realize other communication paradigms such as a shared memory (e.g., [8, 9] or higher communication protocols (e.g., [10]).

3.1.2. Component Interfaces The interfaces of a component abstract from the internals of the component and enable the exchange of one component independently from other components as long as the behavior at the component's interface remains stable. In the GENESYS architecture two types of interfaces of a component are distinguished: local interfaces and linking interfaces (LIFs). Local interfaces are interfaces to the component's environment or in case of gateway components to other subsystems. Linking interfaces are used for the integration of components. A LIF needs to be technology independent in the sense that its specification does not incorporate implementation details of the component. This way it is ensured that different implementations of computational components can be integrated (e.g., general purpose CPU, FPGA, ASIC).

For facilitating composability and interoperability, a precise LIF specification in the temporal and value domain is required [11]. In addition, an interface model assigns meaning to the syntactic structure of the operational specification.

3.2. System Structuring

GENESYS addresses embedded systems design at various levels of integration, which usually also follow different design paradigms. At the lowest level GENESYS considers the closed integration of IP cores interconnected by network-on-chip (NoC), which is typically the top down design of a chip. The other extreme focused by GENESYS is the bottom up ad-hoc integration of of independently developed, autonomous systems (e.g., ambient intelligence). Common to all integration levels is the *waistline* architecture, which means that there is a rather small set of services that needs to be provided by every platform that adheres to the GENESYS architecture. Additional services, which are beneficial for particular domains can be built upon those mandatory services. In the following, these concepts are explained in more detail.

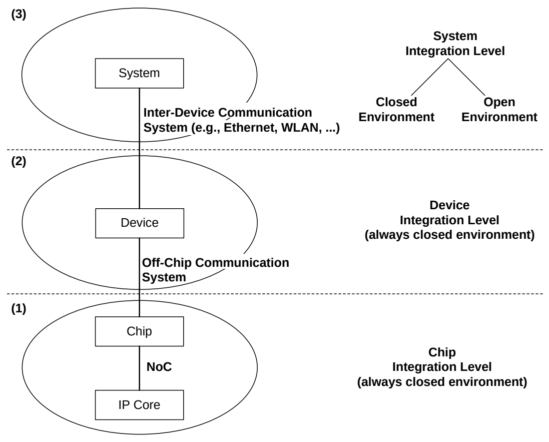
(3)
System
Inter-Device Communication System (e.g., Ethernet, WLAN, ...)
System Integration Level
Closed Environment
Open Environment
(2)
Device
Off-Chip Communication System
Device Integration Level (always closed environment)
(1)
Chip
NoC
IP Core
Chip Integration Level (always closed environment)

Figure 2. GENESYS Integration Levels

3.2.1. Integration Levels Due to the substantial differences of service characteristics at different integration levels, the GENESYS architecture introduces three different integration levels: the chip-level, the device-level, and the system-level (see Figure 2). For instance, the bandwidth in an NoC used at the chip-level is orders of magnitude cheaper than the bandwidth that might be available at a system-level network (e.g., WLAN). This has major consequences on the design decision for the implementation of a communication infrastructure at the different integration levels. Likewise, temporal guaranties for communication activities can only be given in closed systems like systems at the chip-level or device-level.

System Level: A system consists of devices, each of which is a spatially and logically self-contained apparatus (e.g., an ECU in a car or a mobile phone. Devices are connected with each others via their LIFs. This connection can be realized in an open environment or closed environment. In an open environment the composition of devices occurs dynamically during the system operation without a priori knowledge concerning the participating devices. A closed system is a system where all devices are known a priori.

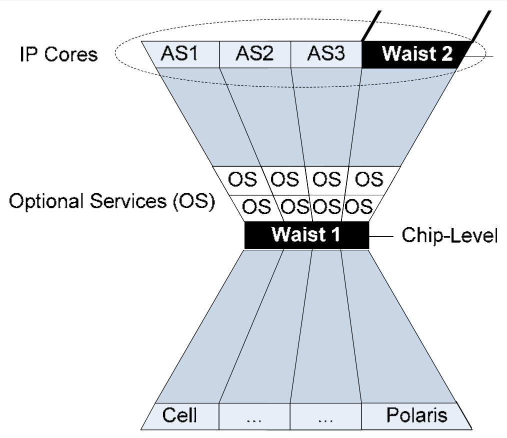

Figure 3. Waistline Architecture (Chip-Level)

Device Level: If a device has an internal structure that is relevant from the point of view of the architecture, then the device can be decomposed into a set of chips that interact via inter-chip LIFs.

Chip Level: If a chip is an MPSoC that has an internal structure with relevance from the point of view of the architecture, then the chip can be decomposed into a set of IP cores. The IP cores communicate with each other using inter-IP core LIFs via NoCs.

It should be noted that not all three levels must be present in every instantiation of the GENESYS architecture. For example, in a single chip system the connection to the open world, e.g. the Internet, could already be at the chip integration level removing the need to consider the device and system integration levels.

3.2.2. Waistline Structure The GENESYS architecture defines generic platform services (e.g., operations to send and receive a message, mechanisms for diagnosis and fault handling) as a foundation for the development of applications. Since the GENESYS architecture follows the principle of a *waistline architecture*, there are two types of platform services at any given level of integration (see Figure 3 for a schematic representation of the waistline at the chip-level):

Core services: The core services are mandatory in every instantiation of the architecture. The core services provide the foundation for higher-level, optional services. For instance, a message transport service is a core service. The core services at the chip-level are denoted as atomic core services.

Optional services: These services build upon the core services and can be generic in the sense that they can be used in multiple application domains or specific for a focused domain. These services are optional in the sense that they are not required in every instantiation of the architecture. For instance an encryption service could be a generic optional service.

The role of the core services and optional services is depicted in Figure 3. At any given integration level, the core services form a waist that can be realized using a multitude of implementation choices. Also, they form the starting point for the domain-customization using optional services. The waist should include exactly those features that are required by all the targeted application domains at the given integration level. Those features that are not used by all application domains can be realized above the waist within the optional platform services. Exemplary categories of core services are networking services, robustness services, and composability services.

3.2.3. Abstraction Levels The GENESYS architecture provides two distinct abstraction levels for the design of embedded systems, namely a logical system view and a physical system view. The purpose of those abstraction levels is to hide implementation details and to provide to the designer only those information that is required in the respective design phase.

The logical system view describes the services provided by the system and their interconnections. In GENESYS terminology a system consists of *jobs* that interact (and provide their services) via message-based interfaces. A job is regarded as an atomic unit and the internals of a job are only relevant for the developer of the job but not for system integration. That means a job can be internally structured into multiple software modules or multi-threaded tasks, but this information is abstracted from the system integrator via the message-based interfaces. A set of jobs that logically form a unit, are grouped together to a *Distributed Application Subsystem (DAS)*. A DAS is a nearly-independent subsystem, that provides a part of the system's overall functionality. The logical structuring of a system into DASs and jobs is independent from the physical location of the jobs in the final embedded system. According to the terminology of the Model Driven Architecture (MDA), the logical system view corresponds to a Platform Independent Model (PIM).

The physical system view describes the concrete implementation of the system. It consists of components, each being a physical entity that can host a job. The three integration levels possess specific types of components, namely IP cores at the chip-level, chips at the device level and devices at the system level. In model-based development according to the MDA, the physical view would be part of a Platform Specific Model (PSM).

4. Commonalities and Contrasts

The development of both architectures, NoTA and GENESYS, are driven by very similar objectives: to tackle the ongoing digital convergence in modern embedded systems and the resultant challenges. This section elaborates on similarities of both architectures, but also points out the major differences.

4.1. Commonalities

4.1.1. Service Orientation In system design, both architectures focus on the identification and specification of services including the specification of the behavior of the services, i.e., the observable interactions at the service interfaces. In terms of GENESYS, this is the identification of internally cohesive subsystems of the overall system, the Distributed Application Subsystems (DASs), and their further decomposition of jobs, each of which is providing a well-defined service. In NoTA this first design step is concerned with the description of the system's functionality by means of service nodes and application nodes, which are logically grouped to subsystems.

The second phase of system design is concerned with a reasonable allocation of those functional entities (jobs in GENESYS, ANs and SNs in NoTA) onto physical hardware entities. In NoTA the term *device* is used to denote a *component*, i.e., the entity that represents the integration of hardware and software to implement a dedicated part of the overall system's functionality. Depending on the level of integration, the terms used in GENESYS are *IP cores* for the chip level, *chips* for the device level, and *devices* for the system level.

4.1.2. Component-Based Design One of the primary aims of both architectures is to reduce time-to-market and costs of future embedded systems and to make the resulting products insensitive to technology changes as much as possible. In order to reach this goal, both architectures push the building of components with explicitly defined interfaces via which the interactions among components are performed. A component is a self-contained subsystem that can be independently developed and used as a building block in the design of a larger system. It is a replaceable part of a system that encapsulates the implementation.

This encapsulation of functionality into dedicated components with well-defined interfaces facilitates the evolvability of the system: For instance, due to the explicit interface specification, components can be exchanged without the need to alter any interacting components as long as the component's behavior at its interfaces remains stable.

4.1.3. Interface Specification In order to achieve the above mentioned benefits of component-based design, a strict interface specification is required. To enable interoperability between components, this interface specification needs to precisely describe the component's behavior at the interface in the value domain and the temporal domain. This operational specification includes functional as well as non functional properties (e.g., dependability properties) and syntactic properties of the exchanged information. In addition, a semantic specification of the interface is required to decide on the interoperability of different component implementations.

The interfaces for the interconnection of components are denoted *Linking Interfaces (LIFs)* in the GENESYS architecture. A precise LIF specification includes the following information: input and output assertions, specification of syntactic, temporal and dependability properties, semantic specification, and periodic ground state (i.e., interface state at the restart instant of a component).

The interface specification in NoTA, denoted Service Interface Specification (SiS), is split in two parts [5]: The *control interface* describes the input and output messages that can be send or received by a service as well as the externally observable states of the service using a protocol state machine. The *data interface* consists of a list of data types each service supports for communication with other services as well as a finite state machine that describes non functional properties (e.g., power consumption) for the data transfer between services.

4.1.4. Stable Architectural Services Both architectures define a stable set of architectural services that can be utilized by application developers. The interface to this architectural services is independent from the actual underlying implementation technology (e.g., transport protocol of the communication infrastructure). This minimizes the migration effort of already implemented services to new technologies.

The architectural services that are available on every instantiation of the architecture are denoted *core services* in GENESYS. Among the core services are basic communication services (periodic, sporadic, and streaming communication), robustness services, security services, and resource management services.

NoTA provides fully specified Application Programming Interfaces (APIs) to ANs and SNs to access the architectural services provided by both layers of the DIP. Whereas the H_IN is mainly responsible for registration, discovery, and activation/deactivation of services, the L_IN provides the functionality for interconnecting components via the actually used physical communication infrastructure. The H_IN is completely independent from the underlying communication technology, parts of the L_IN, on the other hand, need to be ported to a changed technology. The application, however, is unaffected from those changes.

4.1.5. Extensibility The architectural services of both architectures form a stable foundation for application development. However, in particular legacy applications would require a additional functionality (e.g., a particular protocol stack) that exceeds the capabilities of the architectural services. In order to keep the set of architectural services small and stable, both architectures support mechanisms for extensibility.

The waistline architecture of GENESYS is chosen for exactly this reason. While the set of core services remains stable, optional services can be used to extend the architecture and provide higher-level services to application developer. On the other hand, the stable waist of the architecture – the core services – hide changes of the implementation technology from the application.

The design of NoTA does not inherently support this extensibility. But the use of proxy layers upon the H_IN of the DIP enables the provision of higher-level services (e.g., the Khronos protocol stack[3]) upon the NoTA communication infrastructure. Analogous, adapters from the L_IN of the DIP to a particular transport layer (e.g., MIPI[4]) can be used to hide a different implementation technology.

4.2. Differences

4.2.1. Extent of the Architecture NoTA is a framework for the development of application mainly targeted to the needs of the mobile industry. The framework provides the DIP and well-specified interfaces to access the interconnect and use its services. However, apart from the basic functionality of the interconnect no further services are specified. For example NoTA does not intend to define and provide a set of general security services that might be useful in many mobile applications.

The GENESYS architecture goes beyond this core functionality. Within the GENESYS project two major parts of the architecture are developed: The GENESYS architectural style defines architectural principles each instantiation of the GENESYS architecture needs to adhere to, i.e., a guideline how to develop a concrete instantiation of the architecture. The reference architecture template, on the other hand, consists of a concrete set of services: core services as well as optional services. The core services needs to be implemented on each instantiation of the GENESYS architecture, whereas the optional services represent higher-level services that are only of importance of a dedicated application domain (e.g., memory partitioning and protection services for industrial applications).

4.2.2. Guaranties for Architectural Properties The higher layer of the DIP of NoTA, the H_IN, completely abstracts from the implementation technology, i.e., it also abstracts from the transport layer that is used by the lower layer of the interconnect to connect different components. Although functional and non-functional properties of component interactions can be specified by the component's interfaces (e.g., communication latency, energy efficiency), it is hard to guarantee those properties, since they very much depend on the actual implementation of the underlying platform. For instance, if standard Ethernet is used for data transport, a specified maximum transmission latency between components A and B cannot be guaranteed in case a (perhaps faulty) component C monopolizes the communication link to component B.

Also, the GENESYS architecture does not specify a dedicated transport protocol to be used. However, the selection and design of the core services enforce properties of the used transport protocol that ensure that important properties of the architecture can be guaranteed. For instance, a dedicated communication paradigm, the periodic message transport, ensures the timely transport of messages with respect to guaranteed bandwidth, transmission latency and latency jitter. Also, fault isolation between components is provided by encapsulation mechanisms of the GENESYS architecture [7]. On the other hand, the NoTA approach is very flexible: For instance, properties concerning reliability (e.g., message retransmission in case of a transmission error) can be easily added by choosing the appropriate transport layer without the need to change major parts of the architecture implementation.

4.2.3. Multiple Integration Levels For the design and integration of embedded applications with NoTA a single level of abstraction is used: In NoTA multi-

3 `http://www.khronos.org`
4 `http://www.mipi.org`

ple devices, each of which is hosting one or more subsystems, are integrated via the DIP. The internal physical structure of a device is only of minimal relevance for NoTA: The device is required to provide adequate resources for the implementation of the hosted subsystem(s) and if multiple subsystems are located on a single device, the only way subsystems can interact is via the service interfaces. That is, from the logical point of view there is no difference whether two subsystems are located on a single device or on different devices. From a physical point of view, NoTA does not explicitly state at which level the integration of ANs and SNs takes place. So NoTA can be implemented on a single chip, on multiple chips using, e.g., MIPI Unipro for off-chip communication, or on a set of physical devices using WLAN or cellular communication.

GENESYS, on the other hand, rigorously distinguishes multiple integration levels. Similar to NoTA, at each integration level the only way components can interact is via the LIFs of the jobs. At the highest level, the system level, devices are integrated, which are interconnected by an inter-device LIFs. If the internal structure of the devices is of relevance for the system designer, a device can be decomposed into a set of chips, interacting via inter-chip LIFs. If relevant, chips can be further decomposed into a set of IP cores interacting via inter IP-core LIFs. The interface specification is identical at each integration level. This is necessary in order to enable the specification of services in terms of jobs independent from their actual implementation. However, the functional and non-functional properties of the LIFs may differ at the individual integration levels depending on the deployed communication infrastructure.

4.2.4. Focus of the Architecture The origin of NoTA is the mobile industry. Thus, the services provided by the DIP are mainly tailored to the mobile consumer domain (e.g., service registration, discovery and access are integral parts of NoTA). The aim of GENESYS is to represent a cross-domain reference architecture for embedded systems that can be instantiated for different application domains. Hence, in addition to the core services, GENESYS specifies optional services that address the particular needs of the individual domains.

4.2.5. Status Research on NoTA has started in 2003 by the Nokia Research Center. Since then several releases of the implementation of the interconnect have been developed. Since 2007, the results of this development are open to the public including industry as well as developer communities.

The GENESYS architecture is developed in the course of the GENESYS (GENeric Embedded SYStem Platform) FP7-ICT collaborative project that has started on January 1st 2008. Due to the rather short time frame so far, the GENESYS architecture is still in its development phase. So far, a public report describing the Cross-Domain Architectural Style of the GENESYS architecture is available [12] as well as a first version of the GENESYS Reference Architecture Template describing the core and optional services of the GENESYS architecture.

5. Instantiation of NoTA on GENESYS

The similarities between NoTA and GENESYS as discovered in the previous section raise the question, whether it might be possible to instantiate NoTA on top of the GENESYS architecture. It is the purpose of this section to provide a theoretical evaluation of this matter.

5.1. Compatibility in System Structure

The first question to be answered is whether the design methodologies of both architectures match. In NoTA, applications are described in terms of service nodes (SNs) and application nodes (ANs), which interact via strictly defined interfaces. The counterpart hereto are jobs in the GENESYS terminology.

In NoTA, subsystems are used to group SNs and ANs that belong together to larger functional units. Subsystems in the terminology of NoTA are not only logical constructs, but also represent a physical entity that provides adequate resources to implement the hosted services. Hence, subsystems also consist of a physical interface to the physical interconnect. If interacting subsystems are located on the same device, this interface might connect the subsystem to a intra-device interconnect, whereas subsystems located on different devices communicate via a inter-device interconnect.

The GENESYS architecture provides the concept of DASs to perform a logical integration of multiple units to a single (logical) entity. A direct equivalent to the concept of subsystems is not provided by GENESYS, since the issue of physically structuring a system at multiple level of detail is solved in a general way by the different integration levels. At the chip-level, the strict component orientation of the GENESYS architectural styles requires a strict one-to-one mapping of jobs to IP cores. That is, each job is implemented by exactly one IP core. Thus, multiple jobs that form a logical unit and due to their interaction patterns re-

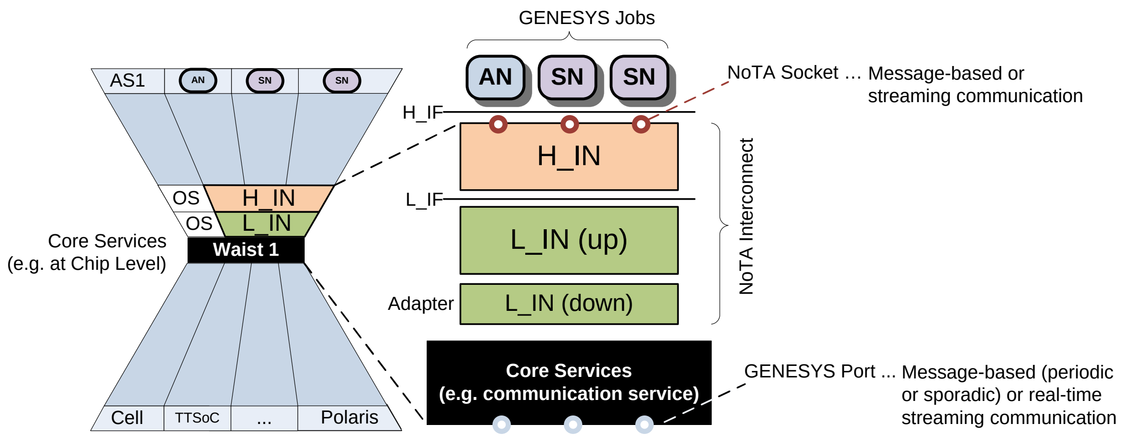
GENESYS Jobs
AS1
AN
SN
SN
AN
SN
SN
NoTA Socket ... Message-based or streaming communication
H_IF
H_IN
OS
H_IN
OS
L_IN
L_IF
L_IN (up)
Core Services (e.g. at Chip Level)
Waist 1
NoTA Interconnect
Adapter
L_IN (down)
Core Services (e.g. communication service)
GENESYS Port ... Message-based (periodic or sporadic) or real-time streaming communication
Cell
TTSoC
...
Polaris

Figure 4. Placing NoTA as Optional Service in the GENESYS Waistline Architecture

quire spatial proximity for an efficient implementation are implemented as IP cores on the same chip. This allows the representation of NoTA subsystems in the GENESYS architecture.

In order to implement the physical interface of a NoTA subsystem, a dedicated gateway IP core can be used in the GENESYS architecture to interconnect each chip to a chip-external network. The next integration level in GENESYS, the device level, provides the encapsulation of several chips into one device. This is analog to the integration of multiple NoTA subsystems into a single NoTA device. The physical interconnection of NoTA devices to the overall system is represented by the system level in GENESYS.

5.2. NoTA as Optional Service on top of GENESYS Core Services

The DIP of NoTA acts as a middleware layer for application development. It provides a stable interface, the interface of the H_IN to the applications. Furthermore, parts of the lower layer of the interconnect, the L_IN, are used to adapt the interconnect implementation to the actual transport protocol. The examples described in [13, 14] uses such adapter to stack NoTA on top of the MIPI protocol. Likewise it is reasonable to build NoTA as optional service on top of the GENESYS core services and to develop an adapter that connects the NoTA interconnect to the core communication services of GENESYS. The example outlined in Figure 4 depicts a GENESYS system at the chip level using, e.g., the Cell architecture [15] or the TTSoC architecture [16] as platform to provide the core services at this level.

The GENESYS architecture seems to be well-suited as a implementation platform for NoTA, since there is already a partial overlapping between the services provided by the DIP and the core services of the GENESYS architecture: The DIP provides message-based communication and streaming communication to the ANs and SNs for realizing their service functionality. Both types of communication are also natively provided by GENESYS (via periodic or sporadic message based communication as well as real-time streaming).

The main extensions of NoTA compared to the GENESYS core services are service registration/discovery and NoTA-specific resource management. In NoTA, a dedicated node, the so-called resource management service node (RMSN), is used to implement this functionality. The RMSN is realized as a dedicated node in one subsystem. The same strategy can be followed when implementing NoTA as optional service for GENESYS, i.e., to use a dedicated job for implementing the functionality of the RMSN.

6. Conclusion

The development of NoTA and Genesys is inspired by very similar challenges. For the solving of those challenges many commonalities between both architectures can be found. The main similarity is the component-orientation and that both architectures take the achievement of composability and interoperability as an objective with top priority. As a consequence, both architectures are based on service specifications which include rigorous specifications of interfaces.

In contrast to GENESYS, NoTA is tailored to applications of the mobile consumer electronic domain. Therefore the NoTA framework does not include the specification of any additional services, which are intended to be used by a broad range of applications across different domains. Hence, the design of the interconnect of NoTA focuses on the needs of the mobile consumer electronics domain (e.g., means for service registration and discovery).

Due to the similarities of both architectures and their non-contradicting architectural concepts, it seems possible to combine both concepts by realizing the NoTA interconnect as optional service on top of the GENESYS core services. This could beneficially influence both architectures: GENESYS by taking advance of a mature architecture from the mobile consumer electronics domain which may strengthen its position in this domain. NoTA by using a platform that is designed to provide fault-tolerance and resilience against transients [7] in order to improve the reliability of the products in this domain.

Acknowledgments

This work has been supported by the European research project GENESYS under project number FP7/213322. The discussions within the GENESYS project and within the research group at the TU Vienna, particularly with Roman Obermaisser, Kari Tiensyrjä, and Eila Niemelä are warmly acknowledged.

References

[1] ARTEMIS. Strategic research agenda - first edition, 2006. At `http://www.artemis-sra.eu/downloads/SRA\_MARS\_2006.pdf`.

[2] ARTEMIS. Reference designs and architectures. constraints and requirements, 2006. At `http://www.artemis-sra.eu/downloads/RAPPORT_RDA.pdf`.

[3] K. Kronlf, S. Kontinen, I. Oliver, and T. Eriksson. A method for mobile terminal platform architecture development. *Advances in Design and Specification Languages for Embedded Systems*, pages 285–300, 2007.

[4] R. Suoranta. New directions in mobile device architectures. *9th EUROMICRO Conference on Digital System Design: Architectures, Methods and Tools (DSD'06)*, pages 17–26, 2006.

[5] J. Lilius, J. Lindqvist, I. Porres, D. Truscan, T. Eriksson, A. Latva-Aho, and J. Rakkola. Testable specifications of nota-based modular embedded systems. TUCS Technical Report No 841, Turku Centre for Computer Science, September 2007.

[6] A. Lappetelinen, J.-M. Tuupola, A. Palin, and T. Eriksson. Networked systems, services and information. the ultimate digital convergence. *First International Network on Terminal Architecture Conference*, 2008.

[7] R. Obermaisser, C. El Salloum, B. Huber, and H. Kopetz. Fundamental design principles for embedded systems: The architectural style of the cross-domain architecture genesys. In *Proceedings of the 12th IEEE Int. Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*. IEEE, 2009.

[8] K. Li and P. Hudak. Memory coherence in shared virtual memory systems. *ACM Transaction of Computer Systems*, 7:321–359, 1989.

[9] B. Fleisch and R.H. Katz. Mirage: A coherent distributed shared memory design. *Proc. of the 12th ACM Symposium on Operating Systems*, pages 211–223, 1989.

[10] Hermann Kopetz, Roman Obermaisser, Philipp Peti, and Neeraj Suri. From a federated to an integrated architecture for dependable embedded real-time systems. Technical Report 22, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2004.

[11] Hermann Kopetz and Neeraj Suri. Compositional design of RT systems: A conceptual basis for specification of linking interfaces. In *Proceedings of the Sixth IEEE Int. Symposium on Object-Oriented Real-Time Distributed Computing*, pages 51–60. IEEE, May 2003.

[12] GENESYS. Cross-domain architectural style. Project Deliverable D6-2, Vienna University of Technology, 2008.

[13] A. Lappetelinen. Extending nota for distributed products, 2008. Slides of the presentation at First International Network on Terminal Architecture Conference.

[14] F. Pourbigharaz and M. Aleksic. Nota imaging solution, 2008. Slides of the presentation at First International Network on Terminal Architecture Conference.

[15] H. Gschwind. The Cell Broadband Engine: Exploiting multiple levels of parallelism in a chip multiprocessor. *International Journal of Parallel Programming*, 35(3), 2007.

[16] Hermann Kopetz, Christian El Salloum, Bernhard Huber, Roman Obermaisser, and Christian Paukovits. Composability in the time-triggered system-on-chip architecture. *21st Annual IEEE International SoC Conference*, Sep. 2008.