Performance Evaluation of Distributed Applications via Kahn Process Networks and ABSOLUT

Suabyal Khan, Jukka Saastamoinen, Jyrki Huusko
VTT Technical research Center of Finland,
FI-90570, Oulu, Finland
e-mail:{subayal.khan,jukka.saastamoinen,
jyrki.huusko }@vtt.fi

Jari Nurmi
Tampere University of Technology,
Department of Computer Systems
P.O. Box 553, Korkeakoulunkatu 1,
FIN-33101 Tampere, FINLAND
jari.nurmi@tut.fi

***Abstract*—The modern mobile devices support diverse distributed applications. The rapid deployment of these applications demands brisk system-level performance evaluation. Abstract workload based performance simulation (ABSOLUT) has been successfully employed to evaluate the performance of non-distributed applications. The main advantages of ABSOLUT include the use of standard tools and languages for example SystemC and UML2.0. To extend ABSOLUT for the system-level performance simulation of distributed applications, application workload models executed in one device must trigger the execution of corresponding workload models in another device. The main contribution of this article is the application of Kahn Process networks (KPN) model of computation (MOC) to extend ABSOLUT for the system-level performance simulation of distributed applications. The approach is experimented with a case study which employs GENESYS application architecture modelling. The GENESYS adopts a service-oriented and component-based design for distributed applications. The approach is not limited to GENESYS and can be used for performance evaluation of distributed applications designed via other application design approaches. The UML2.0 MARTE profile, PapyrusUML2.0 modelling tool and SystemC were used for modelling and simulation.**

Keywords-ABSOLUT; Kahn Process Networks; GENESYS; distributed applications

I. INTRODUCTION

The modern mobile handheld multimedia devices support diverse distributed applications [1]. These applications are often computationally intense and are supported by heterogeneous multiprocessor platforms. The challenges in the deployment of such applications are twofold, i.e., the heterogeneous parallelism in platforms, and the performance constraints.

The abstract workload based performance simulation (ABSOLUT) approach has been extensively applied for the performance simulation of non-distributed applications where all the processes of an application are running on the same platform and communicate via an inter-process communication (IPC) mechanism.

So far, ABSOLUT [2] has not been used to evaluate the performance of distributed applications, where the use-cases span over the multiple devices operating in a ubiquitous environment. In the performance models of such use-cases, the application workload models are mapped to the platform models of the interacting devices. The execution of these workload models must be synchronized to mimic the modelled real world use-case.

A Model of Computation (MOC) is a general way of describing the behaviour of a system in an abstract and conceptual form, enabling the representation of system requirements and constraints at a higher level. In general, the MOC is described in a formal manner via mathematical functions or set-theoretical notations or a combination of them. Kahn Process Network (KPN) MOC is a process based MOC [3]. The following properties make KPN a suitable MOC for the performance modelling of distributed applications [4].

- The processes in KPN MOC execute in parallel and independent of each other. Likewise, in case of distributed applications, the processes hosted by different devices communicate via transport technologies, execute in parallel and are independent of each other.
- Moreover, KPNs are determinate, i.e. irrespective of the employed scheduling policy, for a given input set, the same results will be obtained. This feature of KPN MOC gives a lot of scheduling freedom that can be exploited while mapping process networks over various platforms in the design space exploration process.
- Another desirable feature of KPN MOC is that the control is completely distributed over individual processes and no global scheduler is required. As a result, distributing KPN for execution on a number of processes is simple.
- The exchange of data is via FIFO buffers, i.e. there is no global memory that has to be accessed by multiple processes. In this way, the resource contention is greatly reduced if the systems with distributed memory are considered.
- The synchronization mechanism is implemented via blocking read mechanism on the FIFO channels. It is quite simple and can be efficiently realized in both hardware and software.

The main contribution of this paper is the instantiation of KPN MOC [5] on top of ABSOLUT performance models to extend it for the system-level performance simulation of distributed applications. The approach is experimented with a case study which elaborates the system-level performance

 ISBN: 978-1-61208-171-7

evaluation of distributed GENESYS [5] applications. The approach is also applicable to other distributed applications.

The rest of the paper is organized as follows: Section 2 gives an overview of landmark system level performance simulation techniques and describes the ABSOLUT performance simulation approach briefly. Section 3 provides an overview of GENESYS application architecture modelling. Section 4 explains ABSOLUT performance simulation approach. Section 5 lists the properties of KPN MOC which must be fulfilled for its correct instantiation over ABSOLUT performance models. Section 6 elaborates the main contribution of the article by describing the instantiation of KPN MOC over ABSOLUT for the performance simulation of distributed applications. This section clearly illustrates the way KPN MOC properties mentioned in Section 5 are fulfilled by the modelled components. Section 7 identifies the layers of distributed GENESYS applications and mentions the ABSOLUT application workload model layers corresponding to each layer of the GENESYS application model layer. This section also identifies the ABSOLUT workload models that correspond to processing nodes in KPN MOC. In Section 8, the approach is experimented via a case study. Conclusions and future work is elaborated in Section 9.

II. RELATED WORK

Performance modelling has been approached in different ways. SPADE [7] treats applications and architectures separately via a trace-driven simulation approach. Artemis [8] extends SPADE by involving virtual processors and bounded buffers. The TAPES [9] abstracts functionalities by processing latencies which cover the interaction of associated sub-functions on the architecture without actually running application code. ABSOLUT is a system-level performance simulation approach for embedded systems and employs abstract primitive instructions based workload models and cycle-approximate platform component models.

To extend ABSOLUT for the performance evaluation of distributed applications, the performance model must mimic the execution of the use-case. In case of distributed applications, after some processing in one device; a message is sent to another process hosted by another device (usually called a service request) called a server. After processing the request, the server possibly sends a reply back to the requester (usually called the client). In case of GENESYS and other distributed application architectures, this message passing between processes (hosted by different devices) involves Transport, Data link and Physical layers of the OSI model. Therefore the performance models must use either abstract model of these layers to reduce the modelling effort and increase the simulation scheme or employ some other models of communication between these processes.

The instantiation of Kahn Process Network (KPN) Model of Computation (MOC) over ABSOLUT abstracts out the OSI model layers (Transport, Data link and Physical layers) and the processing nodes (Process workload models in ABSOLUT) pass tokens (mimicking passed messages) to one another via FIFO channels. This enhances the simulation speed and also reduces the modelling effort.

III. GENESYS APPLICATION MODELLING

In GENESYS, compliance of architectural views and concepts across application domains forms basis of the cross-domain architectural style [10]. GENESYS reference architecture template provides core and optional services to application components. The core services are fundamental to any architecture. The optional services, built on top of the core services, can be used in applications across multiple domains.

The modelling process starts by describing a set of views defined in GENESYS that are sufficient for the modelling objective. GENESYS use-case view describes the functionality of a system at a higher abstraction level by means of use-cases. The structural view defines the interface between an application and the sub-systems of the execution platform. This interface describes the core and optional services which the different sub-systems of the underlying platform offer to an application. The syntactical view describes the syntax the servers understand in order to access their services. Sub-systems together with their interfaces (set of services) are conceived as servers that admit different messages from the application (client). The behavioural view reflects the behavioural aspects of an application and its encompassing services.

IV. PERFORMANCE SIMULATION APPROACH

The ABSOLUT performance evaluation approach follows the Y-chart model [11], consisting of application workloads and platform model [2]. After mapping the workloads to the platform, the models are combined for transaction-level performance simulation in SystemC. Based on the simulation results, we can analyse e.g. processor utilization, memory traffic and execution time. The approach enables early performance evaluation, exhibits light modelling effort, allows fast exploration iteration and reuses application and platform models [2] .

A. Application Workload Model

The application workload model has a layered architecture as explained in [2]. The hierarchical structure of the application workload model is shown in Figure 1.

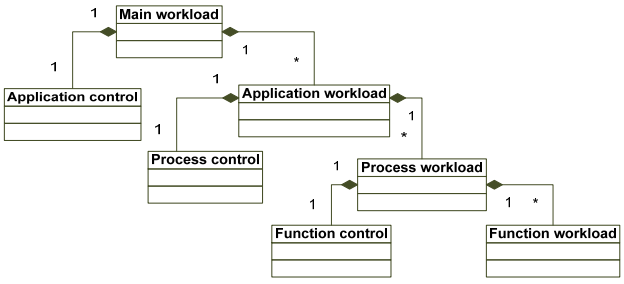

Figure 1: Hierarchical structure of application workload model.

B. *Platform Model*

The platform model is an abstract hierarchical representation of actual platform architecture. It is composed of three layers: component layer, sub-system layer, and platform architecture layer as shown in Figure 9. Each layer has its own services, which are abstract views of the architecture models. Services in sub-system and platform architecture layers are invoked by application workload models [2].

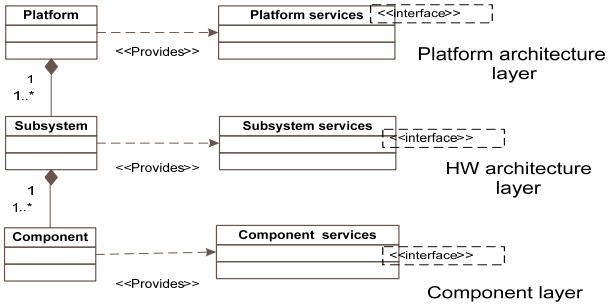

Figure 2: Platform model layers.

V. PROPERTIES OF KPN MOC

KPN [2] consists of nodes that represent processes and the communication channels (unbounded FIFO channels) between these processes. In order to model parallel computation, autonomous computing nodes are connected to each other in a network by communication lines. A given node performs computation on the received data via the input lines using some memory of its own, to produce output on some or all of its input lines. A communication line transmits information within an unpredictable and finite time. At any time a node either computes or waits for information on one of its input lines.

KPN MOC has been used for synchronization among ABSOLUT process workload models running on different platform models (devices in real use-case). In the real use-cases the software components of a distributed application running on different devices pass messages to each other via transport API functions over wired or wireless channels. In the ABSOLUT performance model, this is abstracted by token passing among process workload models over unbounded FIFO channels.

The KPN MOC has a set of properties which must be implemented to ensure the correct instantiation of KPN MOC over ABSOLUT performance models. These properties are listed below.

a. *Read/Write Operations to channels*: The deterministic behavior of KPNs is mainly caused by blocking reads and writes to the FIFO channel instances.
b. *FIFO channel read/write operations*: A process cannot wait for reading/writing of two different FIFO channel instances at the same time.
c. *Channel Access*: If processes can access different FIFO channels and more than one process can run on the same ABSOLUT platform model, then it must be guaranteed that the platform model only allows one process to access a single FIFO channel instance for read/write operation at the same time. This is important since the access to FIFO channels is provided as a service by the ABSOLUT platform (Operating System (OS) model) to the hosted process workload models and more than one process are allowed to access the same platform service.
d. *Process code behavior*: The process code must be blocked of computing while accessing a FIFO channel instance.
e. *FIFO channels*: FIFO channels cannot be active. In SystemC MOC, it means that the FIFO channel models cannot contain sc_threads or sc_methods.
f. *Definition of a KPN processes in ABSOLUT context*: The processes of the KPN network formally correspond to software processes. Therefore either the same convention should be followed or the ABSOLUT workload models corresponding to KPN processes must be identified.

The requirements *a, b, c* and *d* are fulfilled by implementing a general mechanism which allows only one process to access an operating system (OS) service at a time and also blocks the execution of the requesting process until the service request is completely processed. This mechanism is modeled as an OS_Service base class. The Channel Access services, i.e., token transmission and reception are then derived from that base class. These models are explained in Section 6. This section also elaborate the modeling of KPN FIFO channels and the way KPN FIFO channels are accessed, i.e., for read() and write() operations for passing synchronization tokens. Section 7 describes the relationship between ABSOLUT workload models and the KPN processes. Therefore, section 6 and section 7 clearly illustrate the way requirements *a,b,c,d,e* and *f* were satisfied by the modeled components.

VI. INSTANTIATING KPN MOC OVER ABSOLUT

To fulfil the requirements of KPN MOC stated in previous section, the blocking read/write access to FIFO channels, FIFO channels and related services must be implemented and integrated to ABSOLUT.

A. *Implementing Operating System Services*

Instantiating KPN MOC over ABSOLUT demands a mechanism for instantiating new platform services (hardware “HW” and software “SW” platform services). This mechanism is implemented as an OS_Service base class which ensures blocking behaviour and scheduling of the service requests such that only one request is processed at any time for a particular service. The derived services merely implement the functionality making the process of modelling new services straight forward. In this way the

 ISBN: 978-1-61208-171-7

required services are easily implemented by deriving them from the OS_Service base class as shown in Figure 3.

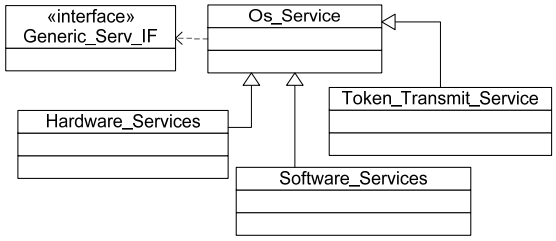

Figure 3: OS_Service base class implementation

The OS_Service class implements the functionality related to scheduling the requests of processes via request queues and informs the requesting process on service completion after taking it to running state again. At one time only one service request is processed. After the processing of a service is completed, the requesting process is informed and then the next request is taken from the front of the request queue for processing. The requesting process is blocked (remains in the sleeping queue of the OS model) until the execution of the request is completed.

More than one processes running on a single platform can request the same service at the same time (in SystemC it means in the same sequence of delta cycles) and are placed in the service request queue of that service. The implementation of the service processing ensures that only one service is processed at a time and when the processing is completed; the next request is fetched for service processing. This is shown in Figure 4.

The three KPN processes (ABSOLUT process-level workload models) running on the same platform (scheduled by the same OS model) and access the platform services via blocking interface are also show in Figure 4. Only one request for a particular service is processed at any time and a process cannot request more than one service at the same time since its execution is blocked until the current request is processed which resulted in blocking its execution.

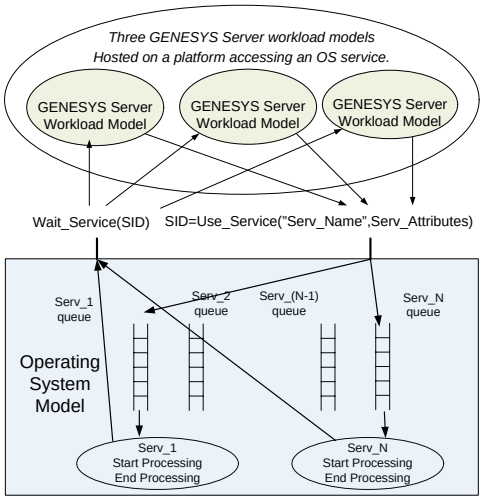

Figure 4: Diagram showing the mechanism employed by OS services to execute requests of processes.

The write access to KPN FIFO channels is implemented as a derived class of OS_Service class and is called "Token_Transmit_Service". This service is registered to the operating system model by the unique service name "TokenTxServ". The Token Passing service is accessed by the Process Workload models by using its unique service name "TokenTxServ" as shown in Figure 5. The blocking nature of this service (blocks the execution of the requesting process) and the scheduling of service requests via queues ensures that the properties *a, b, c* and *d* of the KPN MOC mentioned in Section V are satisfied. The read access to KPN FIFO channels is implemented as a class called "Token_Receive_Service" and is implemented similarly. All the OS_Services are registered to the OS and used via the same interface inside ABSOLUT process workload models. This implementation guarantees that two or more processes cannot access a single FIFO channel instance at the same time. It also guarantees that one process cannot read and write simultaneously at the same time since the execution of the process is blocked until the request is processed.

The read access to the FIFO channels is also implemented in the same way and is a derived class of the OS_Service base class to ensure that the aforementioned properties (*a,b,c and d*) of the KPN MOC are fulfilled.

```
//Access a service named "Serv_Name" with appropriate attributes

Serv_id SID=SERVICE_OS(m_host)->use_service("TokenTxServ",Serv_Attributes);

//Wait for service completion If it is blocking

SERVICE_OS(m_host)->wait_service(SID);
```

Figure 5: Using Token_Transmit_Service by an ABSOLUT process workload model

B. KPN FIFO Channels and Token Modelling

In KPN MOC, the processes communicate via FIFOs channels [2]. If a FIFO channel instance is not empty, the reading is non-blocking. If a FIFO channel instance is empty, the reading process will block.

The standard SystemC 'sc_fifo' channel provides these functionalities [4] and we can use them without any modification. The 'sc_fifo' channel has no sc_threads and no sc_methods and hence is not an active channel since activity in SystemC is only modelled via sc_threads and sc_methods.This fulfils the requirement *e* Mentioned in Section 5.

C. Token Passing and Reception

As shown in Figure 5, while requesting an OS_Service, the requesting process must provide the required service attributes. Therefore for accessing the *"Token_Transmit_Service"* and *"Token_Receive_Service"* services, the requesting processes must provide the required service attributes. The attributes for both these services are modelled as a *"KPN_Token_RW_Attributes"* class which is derived from *"Serv_Attributes"* base class.

 ISBN: 978-1-61208-171-7

The *"KPN_Token_RW_Attributes"* class contains a reference to the FIFO channel to which a KPN MOC Token has to be written or read from. Any two ABSOLUT process workload model communicating with each other (running on different devices "platform models") contain a references to the same FIFO channel instance. One of them can only perform read operations on the FIFO channel instance while the other can only perform write operations.

The Tokens do not represent any data of the real use-case since ABSOLUT employs non-functional application workload models. Therefore tokens are modelled as integer C++ data type, i.e., *int*. The 'KPN_FIFO_Ch' class is derived from 'sc_fifo' primitive channel 'class' and does not contain any additional methods or members. The KPN_Token_RW_Attributes class is shown in Figure 6.

```
class KPN_Token_RW_Attributes : public Serv_Attributes
{

//Other class members
.
.
public:

//channel allocation to this token
void Allocate_KPN_Ch( KPN_FIFO_Ch * param_ KPN_FIFO_Ch){
m_KPN_FIFO_Ch=param_ KPN_FIFO_Ch;
}
.
.
//Channel through which to which this token will be passed
private:
KPN_FIFO_Ch * m_KPN_FIFO_Ch;

//Other class members
.
.
};
```

Figure 6: Attributes class for accessing KPN_Token_Transmit and KPN_Token_Receive service

The aforementioned modelling of KPN FIFO channels and service attributes fulfil the requirement *e* of the KPN MOC mentioned in Section 5.

VII. A KPN PROCESS IN ABSOLUT CONTEXT

Seamless integration of distributed GENESYS application design phase to ABSOLUT application workload modelling phase is conceived as layered application architecture. After defining the layers in the application model, the corresponding layers in the ABSOLUT workload models are identified. In this way, the application model acts as a blue print for the application workload layers [12]. This reduces the time and effort in application workload modelling and speeds up architectural exploration phase.

In GENESYS [10], a distributed use-case can be viewed as a controlled collaboration of service providers and service requesters (both called Servers in GENESYS instead of clients and servers) [5] running on different devices. For example, if the use-case involves the collaboration among "*n*" GENESYS Servers, we can write

$$E_a = \{ C_E, Serv_1, Serv_2, \ldots, Serv_n \}, \quad (1)$$

where C_E represents the control mimicking the collaboration among nodes in order to satisfy the end-user use-case.

In the second layer each GENESYS Server is defined as a process running on a particular platform , i.e.,

$$Serv = \{ P_{GENESYS} \}. \quad (2)$$

where $P_{GENESYS}$ represents a GENESYS Server running on a particular platform(called sub-system in GENESYS).

In the third layer each running GENESYS server ($P_{GENESYS}$) is represented as a controlled invocation of one or more function workload models or platform service requests. If a process consists of "*k*" processes and "*l*" platform service requests, we can write

$$P_{GENESYS} = \{C_P, F_1, F_2, \ldots, F_k, R_1, R_2, \ldots, R_l\}, \quad (3)$$

where C_P is control.

The aforementioned GENESYS application model layers are then compared to the ABSOLUT application workload model layers as shown in Table 1.

TABLE 1: COMPARING GENESYS APPLICATION ARCHITECTURE LAYERS TO ABSOLUT WORKLOAD MODEL LAYERS

GENESYS Application architecture layers	**Corresponding ABSOLUT Workload Model Layers**
$E_a=\{C_A, Serv_1, Serv_2, .., Serv_n\}$	$W=\{C_A, Servwld_1, ,\ldots, Servwld_n\}$
$Serv = \{ P_{GENESYS} \}$	$Servwld = \{ P_{GENESYSwld} \}$
$P_{GENESYS}=\{C_P, F_1, F_2, \ldots, F_k, R_1, R_2, \ldots, R_l\}$	$P_{GENESYSwld}=\{C_P, Fwld_1, Fwld_2, .. Fwld_k, Rwld_1, Rwld_2, .., Rwld_l\}$

Where Servwld is an ABSOLUT application workload model, $P_{GENESYSwld}$ is an ABSOLUT process workload model and Fwld is an ABSOLUT function workload model. Each ABSOLUT process workload model corresponds to a KPN process which invokes the ABSOLUT function workload models and platform services in a deterministic order.

Each GENESYS Server Application level workload model instantiates the single process workload model mimicking the execution of a GENESYS server of a distributed GENESYS application in the real use-case. Each process workload model ($P_{GENESYSwld}$ in Table 1) corresponds to a single KPN processing node in KPN MOC since the ABSOLUT processes code behaves in the same way as the rules stated in Section 5 by using the "Token_Transmit_Service" and "Token_Receive_Service"

 ISBN: 978-1-61208-171-7

for FIFO channel access. This observation fulfills the requirement *f* of the KPN MOC mentioned Section V.

Therefore from KPN MOC viewpoint, each Application level workload model instantiates a computing node (in ABSOLUT it means a Process-Level Workload models which are shown in blue color in Figure **7**) of the KPN MOC. These computing nodes are connected via unbounded FIFO channels for passing tokens in order to ensure deterministic behavior as shown in Figure **7**. Also one or more Process workload models can run on the same platform as in real use-cases as shown in Figure 7. In Figure **7**, two nodes are running on the same platform (Platform 2). Since the access to FIFO channels is blocking and only one process can read or write to a single FIFO instance at a time as explained in Section 6, therefore the deterministic behavior of KPN MOC is guaranteed.

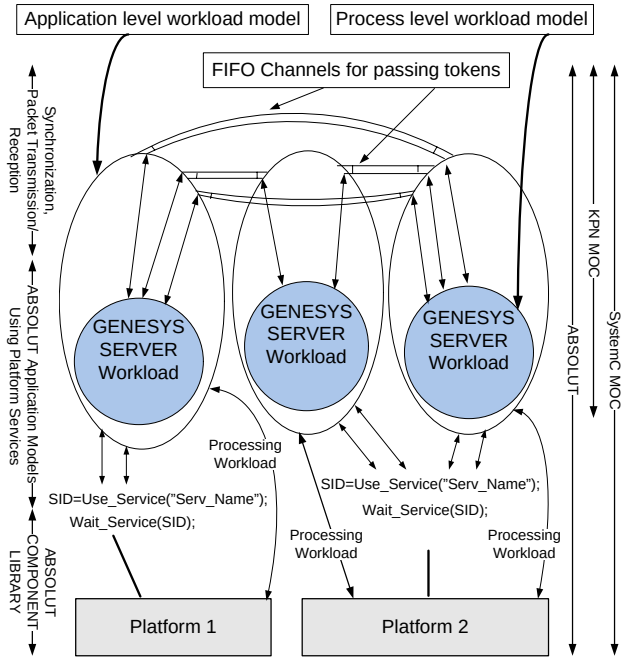

Figure 7: KPN Nodes internals and access to KNP FIFO channels for token passing.

VIII. CASE STUDY

The case study describes the modeling and performance evaluation of an Office Security (OS) application hosted on a mobile device owned by a member of security staff. The application has been previously presented in [7]. The device hosting the application, communicates with three other devices to avail different services. The PersonCounterSubSystem gives the number of occupants in the office and provides the video of office entrance. The OfficeVideoSubSystem provides high resolution office video. The FaceTrackerSubSystem provides the information about the number of occupants sitting on the bench and video showing the occupants.

A. Application Model

The OS application uses services provided by Servers running on different devices. Each Server communicates with its respective streaming Server. Each device hosting a Streaming server is fitted with an integrated camera mounted at an appropriate position in office. The streaming servers stream video frames to the requesting servers on demand as shown in application views. It should be noted that in GENESYS terminology, both the service requester and provider are termed as Servers instead of client and server as in case of internet applications.

1) Use-case view

The use-case view shows a system-level capability SelectTheSecurityService shown in Figure 8 in terms of device services.

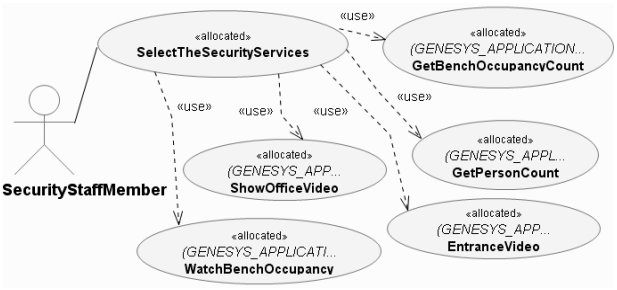

Figure 8: Use-case view.

2) Behavioral View

The Behavioral view shows the behavior of an application. Application invokes different services as use-case evolves. This is shown in Figure 9.

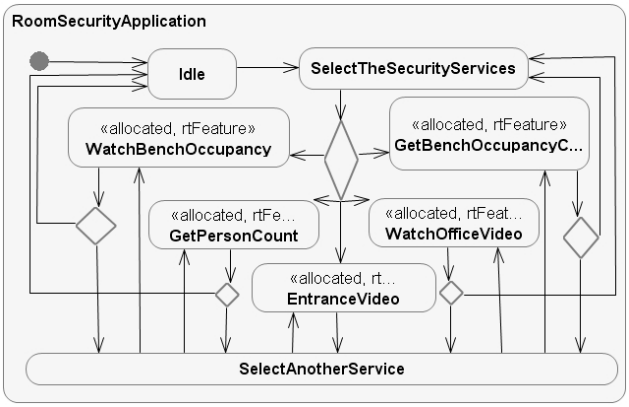

Figure 9: Operation of the Office Security Application.

3) Syntactical View

The syntactical view shows messages admitted by the sub-systems as shown in Figure 10 and Figure 11. The stereotypes used in syntactical view are described in detail in [5].

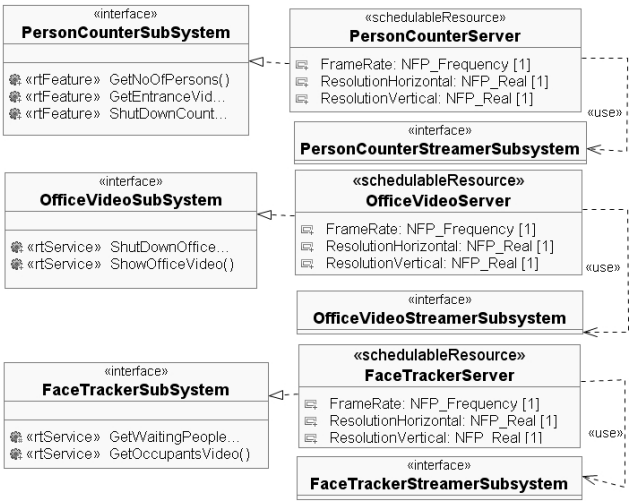

Figure 10: Sub-systems which serve the application directly.

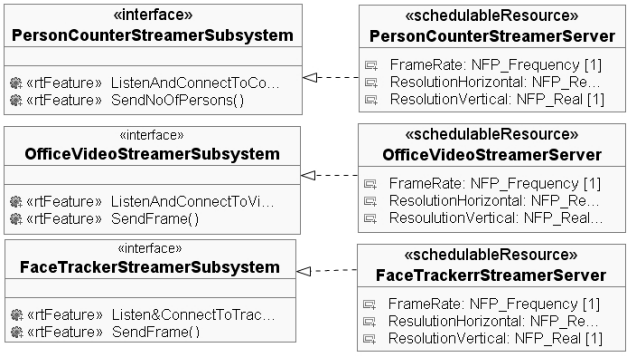

Figure 11: Sub-systems which serve the application indirectly.

B. *ABSOLUT performance model*

In the case of non-distributed applications, the overall performance model consists of a single platform model to which one or more applications workload models are mapped. These application models represent the processing load of the whole use-case [2].

In case of distributed applications, each server and client (both called Servers in GENESYS) in real use-case is modeled as a separate application-level workload model. Each application-level workload model of a GENESYS Server instantiates the process workload model mimicking application execution in the real use-case. In the case of performance models of distributed applications, at least two process workload models are hosted on different platform instances. The process workload models hosted on different platform instances communicate with one another via FIFO channels. The blocking read/write access to the channels is implemented as platform services as explained in Section 6.

The process workload models hosted on same platform communicate with one another via inter-process communication (IPC) ABSOLUT model [13].

Therefore, in case of distributed applications, the overall performance model consists of more than one application-level workload models of clients and servers (both called Servers in GENESYS) hosted by at least two different platform model instances. In this case, the performance results for all the platform models are obtained separately and analyzed to perform optimizations if required.

In the case study, each GENESYS server presented in the application model is mapped to a separate multi-core based platform model to analyze the performance results and identify the potential bottlenecks at the software and hardware side. The KPN view of the performance simulation model is shown in Figure 12. The direction of arrows indicates the direction in which the tokens are passed.

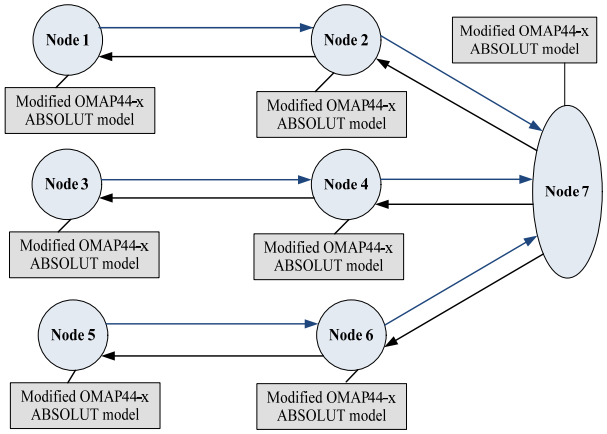

Figure 12: Kahn process network model of the performance simulation model

Node 1 and Node 2 represent the PersonCounter and the PersonCounterStreamer server. Node 3 and Node 4 represent the OfficeVideo and OfficeVideoStreamer server whereas Node 5 and Node 6 represent the FaceTracker and FaceTrackerStreamer server. Node 7 represents the application which is in the form of a control [12].

1) *ABSOLUT Platform Model*

Each ABSOLUT platform model used in the case study is a modified OMAP-44x platform model which consists of a single ARM Cortex-A9 multi-core processor [14] model consisting of four cores along with SDRAM, a POWERVR SGX40 graphics accelerator and an Image signal processor. This is shown in Figure 13. These component models are connected via an AMBA bus model [15].

In ABSOLUT methodology, the application models contain approximate timing information. Thus the execution platform is modelled at transaction level following OSCI TLM2.0 standard [2].

The application workload models do not include accurate address information and therefore the cache architecture is simplified and cache misses are modelled statistically [2]. Processor performance is taken into account by defining clock frequency of cores. Architecture efficiency of cores is modelled as average cycles-per-instruction (CPI) value.

 ISBN: 978-1-61208-171-7

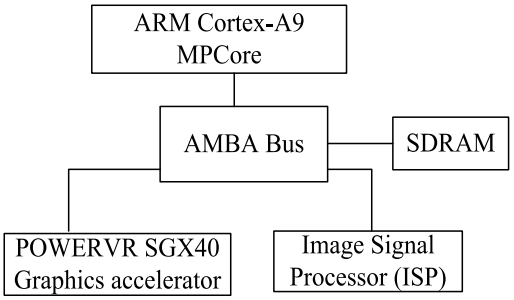

Figure 13: ABSOLUT platform model

Each core of ARM Cortex-A9 MP Core model has an L1 instruction and L1 data cache as shown in Figure 14**.**

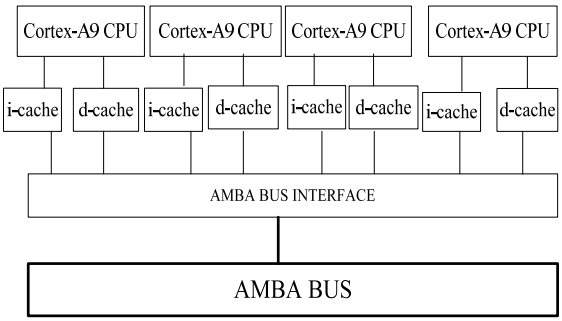

Figure 14: Diagram showing the quad-core processor (ARM Cortex-A9 multi-core processor) model used in the performance

2) Application Workload Model

All the Servers elaborated in the application model were programmed using OpenCV library [16]. The tool used for the workload extraction is ABSINTH [2]. ABSINTH generates one Function workload models for each function in the application if the function lies in the user-space code or is provided by an external library. The workload models of the Transport API functions (mostly system calls) such as TCP/IP, Bluetooth and UDP API functions are not extracted and instead one stub is generated for each system call.

The stubs corresponding to message sending and receiving API functions, for example send() and receive() API function calls in case of TCP/IP are replaced by the service requests of "Token_Transmit_Service" and "Token_Receive_Service" mimicking the transmission and reception of actual message in real use-case. Any two nodes communicating in this way have the reference to the same KPN FIFO channel instance.

C. Co-Simulation and performance results

During the execution of application, the end-user requests the bench occupancy and the video of the occupants. The video frames are streamed form the FaceTrackerServer to the mobile device of the Security Staff member. Then the security staff member invokes other services one by one, switching between them after 1→2 minutes each.

Each GENESYS Server Application workload model is mapped to its respective platforms as shown in Figure 12 and the resultant performance model is run to obtain performance results. The results of all the platforms (called Sub-systems in GENESYS) and their hosted Servers are written to one text file in the form of different sections, one for each platform and its hosted GNESYS servers. In this paper we only present the performance results of the platform hosting the FaceTrackerStreamerServer.

The simulation execution can be easily exited after any pre-decided simulation time, for example after 20 seconds (time in terms of SystemC time model) or another event in the simulation for example the number of streamed packets from one Server to another or from a Server to the Application. When the pre-decided condition is met during simulation, the sc_stop() function is called. After that the destructor of the results reporting class is called which writes the gathered results to a text file for analysis.

1) Performance Results (Platform)

Since the FaceTrackerStreamerServer was implemented entirely as software, the Graphics Accelerator and Image Processor Services available from the platform were not used. Therefore only the utilization of the processor cores of platform hosting FaceTrackerStreamerServer is shown in Figure 15. This platform is called FaceTrackerStreamer-Sub-system as shown in Figure 11**.** The simulation was run for streaming of 10, 100 and 1000 packets. The solid bar corresponds to 10 packets, bar with horizontal pattern shows use-case of 100 packets and diagonal pattern corresponds to 1000 packets.

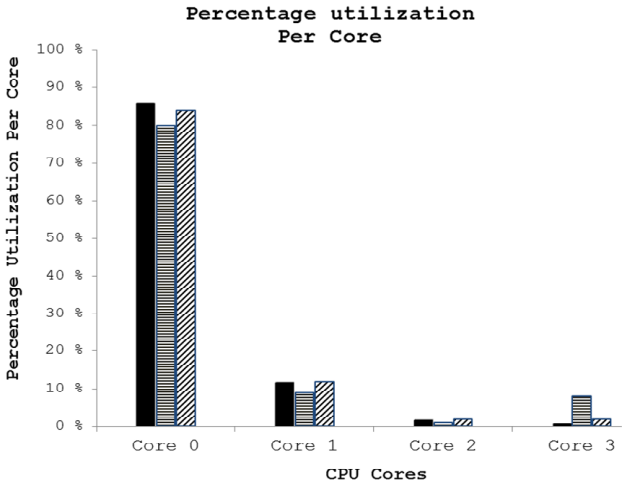

Figure 15: Utilization time of processor cores as compared to overall Utilization time of the CPU

The cache statistics of the platform (FaceTracker-StreamerSubsystem) are shown below for 1000 video frame transmissions.

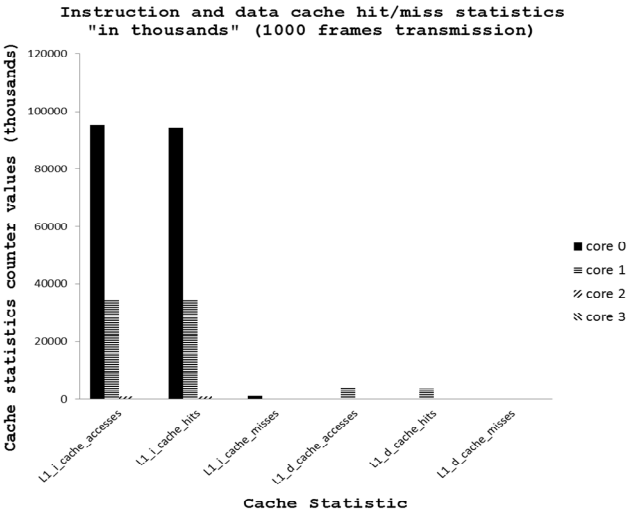

Figure 16: Cache hits miss statistics of the Sub-system (platform hosting a server, i.e., FaceTrackerStreamerServer).

2) Performance Results (Application)

By analysing the processing times of the application source code and the percentage utilization of multi-core processor model by different external library and user-space code, we can find the potential bottlenecks in the application implementation, which will help to perform required optimizations. For example, by analysing the processing times of the functionalities which can affect a particular non-functional property for example frame rate for FaceTrackerStreamerServer, we can find out whether the implementation of the software components satisfies this non-functional property.

This non-functional property is annotated in the application syntactical view. The processing times of the functionalities which can affect the Frame Rate are first identified. In the next step, their processing times and processor utilization percentage with respect to the overall utilization by all application software components are recorded. The functionalities and the corresponding OPENcv library [16] and user space functions which provide these functionalities are shown in Table 2. DetectAndDrawFaces and SendImage are user space functions. SendImage is a wrapper around BSD API send() function.

TABLE 2: SHORTLISTED FUNCTIONS THAT CAN AFFECT THE FRAME RATE (A NON-FUNCTIONAL PROPERTY) OF FACETRACKERSTREAMERSERVER

Functionality	Shortlisted Function
Use Camera for frame capture	cvCaptureFromCAM
Get a frame from camera	cvQuerryFrame
Create and store Image	cvCreateImage
Detect and draw faces	DetectAndDrawFaces
Show the result	cvShowImage
Send the Image	SendImage

The operations mentioned in the above table are pipelined (except cvCaptureFromCAM which is called just once) and are executed in the order shown in the activity diagram of Figure 17.

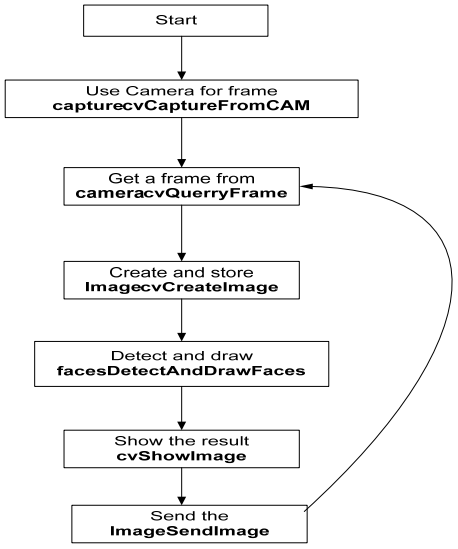

Figure 17: Functionalities and related OPENcv functions that can affect non-functional property Frame Rate

In order to satisfy the required frame rate of FaceTrackerStreamerServer, i.e., 25 frames/sec, each of these operations must be performed within 1/25 seconds (40 milliseconds) [6]. The processing times and the percentage processor utilization of the aforementioned functions are shown in Figure 18 and Figure 19**.** It is seen that all the operations are performed within 22000 microseconds or 22 milliseconds.

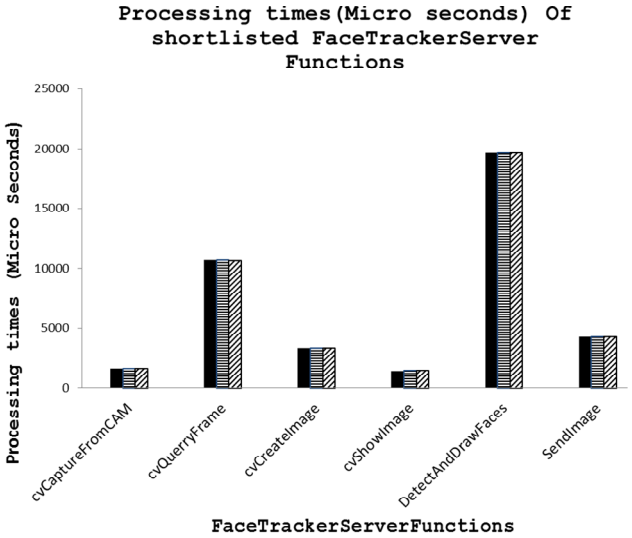

Figure 18: Execution times of functionalities that can affect the non-functional property Frame rate of Face Tracker Streamer Sub-system

The processor utilization graph shows that drawing and detection of faces takes 41% of the CPU time in proportion

to the overall CPU time taken by all the application functions considered.

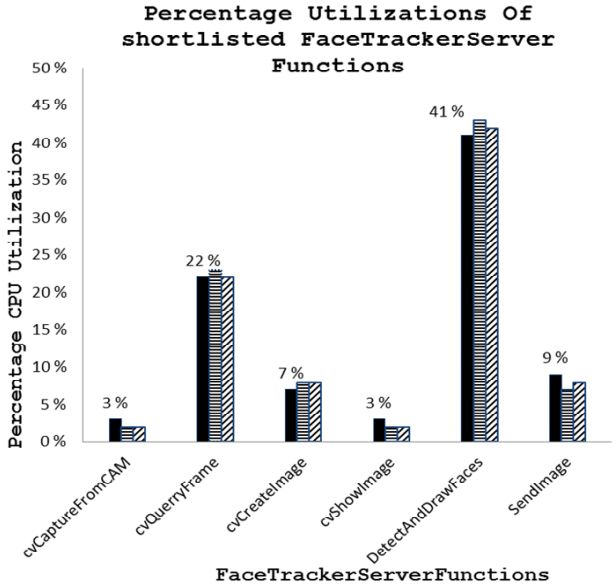

Figure 19: Percentage CPU Utilization of the functionalities that can affect Frame rate of Face Tracker Streamer Sub-system

The obtained performance results are used to perform appropriate changes in the application models by replacing the software components with more light weight implementations or by making changes in the platform model if the performance requirements (non-functional properties) are not met. If the performance requirements are met by all the platform and software components, the architectural exploration stops and the implementation phase starts.

IX. CONCLUSION AND FUTURE WORK

The KPN MOC was instantiated over ABSOLUT performance models to extend it for the performance evaluation of distributed applications. The approach was experimented with a case study. In the future functional MAC, transport and physical layer models will be integrated to ABSOLUT along with active channel models.

ACKNOWLEDGEMENT

This work was performed the Artemis SOFIA project partially funded by Tekes and the European Union. The authors would like to thank also the colleagues at VTT and Future Internet program of TIVIT (Finnish Strategic Centre for Science, Technology and Innovation in the field of ICT) for discussion and support.

REFERENCES

[1] T. Noergaard. Embedded Systems Architecture. A Comprehensive Guide for Engineers and Programmers. ELSEVIER, UK; 2005, 640 p.

[2] Jari Kreku, Mika Hoppari, Tuomo Kestila, Yang Qu, Juha-Pekka Soininen, and Kari Tiensyrja. Combining UML2 and SystemC Application Platform Modelling for Performance Evaluation of Real-Time Systems, EURASIP Journal on Embedded Systems, volume 2008, ARTICLE ID 712329.

[3] G. Kahn. The Semantics of a simple Language for Parallel Programming. Proc. of the IFIP Congress 74, North-Holland, 1974.

[4] F. Herrera, P. Sánchez, and E. Villar. Modeling of CSP, KPN and SR Systems with SystemC. In Proc. FDL, 2003, pp.572-583.

[5] Valentina Zadrija. Survey of Formal Models of Computation for Multi-Core Systems. Technical Report 03/31/2009, Department of Electronics, Microelectronics, Computer and Intelligent Systems, Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia.

[6] Subayal Khan and Kari Tiensyrjä. Instantiating GENESYS Application Architecture Modeling via UML 2.0 constructs and MARTE Profile. In Proceedings of 13th Euromicro Conference on Digital System Design (2010), September 1-3, Lille, France, 2010.

[7] P. Lieverse, P. van der Wolf, K. Vissers, and E. Deprettere. A methodology for architecture exploration of heterogeneous signal processing systems. Kluwer Journal of VLSI Signal Processing 29 (3), 2001, pp. 197-207.

[8] A. Pimentel and C. Erbas. A Systematic Approach to Exploring Embedded System Architectures at Multiple Abstraction Levels. IEEE Transactions on Computers, vol. 55, no. 2, Feb. 2006, pp.99 – 112.

[9] T. Wild, A. Herkersdorf and G.-Y. Lee. TAPES—Trace-based architecture performance evaluation with SystemC. Design Automation for Embedded Systems, Vol. 10, Numbers 2-3, Special Issue on SystemC-based System Modeling, Verification and Synthesis, 2006, pp 157-179.

[10] http://www.genesys-platform.eu/genesys_book.pdf

[11] B. Kienhuis, E. Deprettere, K. Vissers, and P. van der Wolf. Approach for quantitative analysis of application-specific dataflow architectures. The IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP '97), pp. 338–349, Zurich, Switzerland. July 1997.

[12] Subayal Khan, Susanna Pantsar-Syväniemi, Jari Kreku, Kari Tiensyrjä, and Juha-Pekka Soininen. Linking GENESYS Application Architecture Modelling with Platform Performance Simulation. In Proceedings of the 12th Forum on Specification and Design Languages (FDL 2009), September 22-24, Sophia Antipolis, France, 2009.

[13] Jukka Saastamoinen, Khan, Subayal, Tiensyrjä, Kari and Tapio Taipale. Multi-threading support for system-level performance simulation of multi-core architectures. 24th International Conference on Architecture of Computing Systems, ARCS 2011. 02/22/2011-02/23/2011. Como, Italy.

[14] Texas instruments. Retrieved June 01, 2011 from www.ti.com

[15] Transaction-Level Models for AMBA Bus Architecture Using SystemC 2.0.M. Caldari, M. Conti, M. Coppola, S. Curaba, L. Pieralisi, C. Turchetti. 2003 Design, Automation and Test in Europe Conference and Exposition (DATE 2003), 3-7 March 2003, Munich, Germany.

[16] OpenCV (Open Source Computer Vision) library. Retrieved June 11, 2011 from http://opencv.willowgarage.com/wiki/

 ISBN: 978-1-61208-171-7