Towards biologically inspired decentralized self-adaptive OS services for distributed Reconfigurable System on Chip (RSoC)

Sufyan Samara[1], Dalimir Orfanus[2], Peter Janacik[2]
Heinz Nixdorf Institute[1], International Graduate School [2]
University of Paderborn
Paderborn, Germany
sufyan@mail.uni-paderborn.de [1]
{orfanus, pjanacik}@uni-paderborn.de[2]

ABSTRACT

Distributed RSoCs are the next step towards a new generation of embedded systems. Applications running on heterogeneous distributed RSoCs require an OS which dynamically adapts to their variable demands. In this paper, we present a novel decentralized OS service design, which enables OS adaptiveness, resource sharing, and reconfigurability on distributed RSoCs. The challenges faced by this design are classified and discussed. To cope with them, biologically inspired algorithms, e.g. for service discovery, are adopted, which use only local information provided by an RSoC and its direct neighbors.

1. INTRODUCTION

The complexity of embedded systems is increasing continuously. Examples include the multiple systems combined on a single chip such as a Field Programmable Gate Array (FPGA) with a General Purpose Processor (GPP) contained inside it. This forms what is called a Reconfigurable System on Chip (RSoC), e.g. Xilinx Virtex^{TM} II pro.
Distributing such systems expands the complexity to a level where an Operating System (OS) becomes a necessity. An OS provides transparency and resource management (e.g. power). However, in a limited resources embedded system, an OS may also be considered as an overhead. On such systems, an OS needs to be aware of the limited resources and adapt to the variable change of resource availability. Moreover, new design methodologies are needed for OS services running on distributed RSoCs. An OS service can support the utilization of the reconfigurable property found on RSoC and when needed, to be able to migrate fully or partially to execute on FPGA. Doing such a migration at runtime allows an OS to be aware of dynamic resource changes expected in a distributed RSoCs. Other properties, such as self-healing and self-organizing, need to be also supported in the new design.

A centralized novel OS service design for distributed RSoCs was presented in our previous work [1]. Nevertheless, due to centralization, the proposed OS suffers from the problem of a single point of failure. Moreover, to retain the OS, a resourceful RSoC is needed. To work around such issues without losing flexibility and additivity, we propose to distribute the blocks of an OS service across several RSoCs with fully decentralized control, as described in the following sections. This allows dynamic resource sharing and increases the failure tolerance and adaptiveness. This paper presents a novel approach for a fully decentralized OS service management distributed in a network of RSoCs. The approach is based on the *emergent self–organization* metaphor.

This paper is organized as follows: Section 2 introduces the motivation, the adopted design, and the proposed solution. In Section 3 we discuss related work, and finally we provide the conclusion and the intended future work in Section 4.

2. MOTIVATION AND DESIGN

An OS service is designed to support resource awareness, distribution over RSoCs, adaptability, and reconfiguration. For the sake of that, each OS service exists in two implementations: (i) one for FPGA, and (ii) another for GPP. Further, each implementation of a service is partitioned into smaller blocks, called *Small Execution Segments* (SES), see Figure 1.

All SESs with the same index/level of a service in the different implementations have the same behavior, i.e. for a given input they produce the same data at output. Produced output data from one SES can be then transfered into subsequent SES which can be either FPGA or GPP. This gives us approximately 2^n possibilities to execute a service, where n is the number of SESs in a service implementation. Moreover, each SES is assumed to have information about itself. This includes: (i) worst case execution time, (ii) worst case power consumption, and (iii) area allocation, where area represents either the needed number of Look Up Tables (LUT) in an FPGA or the GPP payload/utilization.

In [1] we devised an algorithm which calculates the best suited service configurations on a critical resource RSoC.

The algorithm assumes that all OS services are designed and segmented into SESs which are stored in a repository called *OS Services Repository* (OSR). Each RSoC in the distributed system can connect and request a service from the OSR. The algorithm, residing in the OSR, uses runtime information provided by the requesting RSoC. The algorithm calculates and sends a service configuration which is suitable for the current RSoC demands and constraints. The sent configuration is executed completely using the requesting RSoC's resources. Moreover, each RSoC has a middleware which enables inter-SESs communication and runtime resource monitoring.

In this work we decentralize the SESs across the RSoCs in the network. This is to avoid problems such as a single point of failure. Further this is done to allow more flexibility, adaptiveness, and resource sharing.

In order to realize decentralization, many challenges are to be met. These challenges can be classified into four categories:(i) the initialization stage, (ii) the SESs discovery and execution stage, (iii) the optimization stage, and (iv) the self-organization stage.

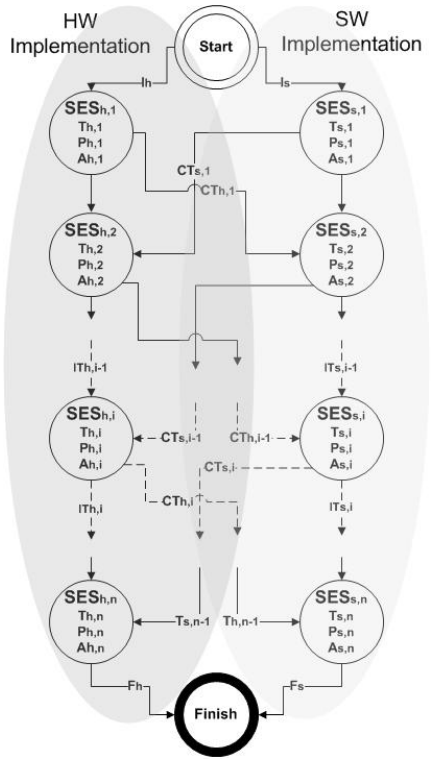

Figure 1: A service SESs implementation

2.1 Initialization

This is the first phase to be realized. A resource heterogeneity is assumed among RSoCs. This raises the challenge of how to distribute the SESs. A repository is assumed in the initial stage to hold the services existing as SESs implementations, see Figure 2. The SESs are distributed randomly on RSoCs. However, if an RSoC receives an SES which exceeds its SES resource reservoir, the SES is rejected and it is propagated to another RSoC. The RSoC SES reservoir is a resource percentage reserved for the sake of storing and running a number of SESs. To allow a kind of balancing, a simple algorithm which involves an updated polarity number and a simple counter is used. The counter in the repository

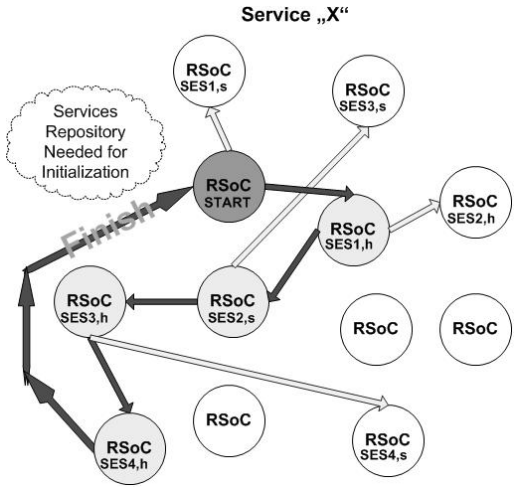

Figure 2: Service discovery and execution example

is initialized to the number of RSoCs and the polarity of each RSoC is set to be positive. All the SESs are with negative polarity. Each time an SES is sent out from the repository, the counter is decreased. If an RSoC with positive polarity receives an SES, it attracts the SES, stores the SES in its resource reservoir, and changes its polarity to negative. On the other hand, if an RSoC with negative polarity receives an SES, it repels the SES to other RSoCs. An SES is propagated through the RSoCs until it reaches an RSoC with positive polarity. Once the counter in the repository reaches zero, a message is sent to reset the polarities of the RSoCs which still have some free resource reservoir to positive, the counter is set to the number of these RSoCs, and then the operation repeats.

2.2 Discovery and Execution

This stage is initialized whenever a service is needed to be executed. For example, in Figure 2 the RSoC denoted by *"START"* will initiate the execution of a service *"X"*. This is done by sending the data to the RSoC containing the first SES of the service *"X"*. For the sake of discovering and executing SESs of one service, we developed a novel biologically inspired *segment discovery algorithm*. The goal of this algorithm is to find in the network the closest suitable implementation for a given SES. To achieve this goal only local information from the neighbors can be used.

Our solution is based on a hop–by–hop reactive routing algorithm [2] which was inspired by the swarm intelligence behavior of ants communicating by changes of their environment [3]. When a food source is found, a chemical substance called *pheromone* is deposited by ants on the paths towards this source. Pheromone can be smelled by others ants and may attract them to the food source. More concretely, during the probabilistic route selection, a route with a higher pheromone value is chosen with a higher probability. Given

its physical properties, pheromone evaporates exponentially with time unless a new portion of pheromone is deposited. If a path becomes favored, there will be a higher amount of pheromone deposited contrary to a path which is less favored. Paths which are obsolete vanish as the last amount of pheromone has evaporated. This particular behavior of ants is desirable in terms of two problems which we face in our distributed RSoCs: (i) Finding the shortest path to the SES, and (ii) finding a good implementation of the given SES. This is achieved *without* any global knowledge of the topology or central coordination.

In our case, each RSoC stores digital pheromones in a structure called *segment routing table* which has the following format:

> destination segment SES_{id} | next hop ID | implementation im | pheromone $\psi_{SES_{id}}$

Every segment has assigned its identifier SES_{id} which is identical for all instances of the segment, independently of the implementation. With every segment is associated ID of the next hop RSoC and also an amount of pheromone on that path to $\psi_{SES_{id}}$ which has been deposited so far, as well as, the type of implementation which may either be h or s. Regularly, every time interval τ, values of pheromones in the table are decreased by pheromone decay coefficient $\psi_\tau \in [0, 1)$:

$$\psi_{SES_{id}}(t + \tau) = \psi_{SES_{id}}(t) \cdot \psi_\tau \tag{1}$$

On a given path, if there is no pheromone deposited for a longer period of time, this path becomes less favorable. In order to avoid the creation of loops, each RSoC has to maintain a *recent message table* saving all identifiers (which are unique) of all messages received according to a least-recently-used strategy.

The segment discovery algorithm consists of two steps: (i) Availability check of a segment and (ii) migration of data to the found SES.

2.2.1 Availability Check

The goal of this step is to check and ensure that there exists at least one path to a requested segment. This is done in two phases: a forward and a backward phase. In the forwards phase, links towards the requester are strengthened, or established respectively. The backward phase does the same towards the RSoC with a requested segment.

Once an RSoC finishes one SES computation, the RSoC migrates data to another RSoC which is hosting the next subsequent segment. For this reason, the RSoC creates a message called *FDAnt* (forward discovery ant) (Figure 3).

When an RSoC receives the FDAnt message it proceeds as follows:

1. RSoC checks: (i) whether the message exceeds its maximum lifetime, or (ii) the RSoC is already in the *History* field. In those cases, the massage is discarded. If the message is contained in the recent messages table, before it is discarded, the message's last hop is recorded in the routing table.
2. If the message is not discarded in the first step, then it follows:
   - The message is registered in all relevant tables.
   - If the RSoC contains required SES, answer with a BDAnt.
   - Propagate the message using broadcast. Unicast is used if relevant pheromone values are exceptionally high. Moreover, it is used in general for all subsequent messages which transfer payload data between the two endpoints.

Field	**Value**	**Field**	**Value**
Last hop	Requestor ID	Last hop	SES_{ID} provider
Hops	0	Hops	0
Type	FDant	Type	BDant
Requestor	ID	Requestor	ID
Requsted SES	SES_{ID}	Requsted SES	SES_{ID}
History	Cleared	History	Cleared
Implementation	h or s	Implementation	h or s

Figure 3: Fields of FDAnt and BDAnt messages

A RSoC which can provide the required segment, i.e. SES_{id}, creates a BDAnt (backward discovery ant) message based on the copy of the received FDAnt. In the backward phase, the BDAnt message (Figure 3) is routed back using links that have been set up by the forward phase.

Unicast routing works as follows: A message m, from origin D_{orig} comes from node h, arriving at node i. Before forwarding it further, i alters the corresponding routing table entries:

$$\psi_{D_{orig,h}} = \psi_{init} \tag{2}$$

if $\psi_{D_{orig,h}} < \psi_{init}$, with ψ_{init} being a value used for initialization. Basically, when there is no pheromone or it is below the threshold of ψ_{init}, it is reinitialized to this level. Else, if $\psi_{D_{orig,h}} \geq \psi_{init}$,

$$\psi_{D_{orig,h}} = \psi_{D_{orig,h}} + \psi_\delta \tag{3}$$

so that a constant amount (ψ_δ) per used link is added to the already existing pheromone level ($\psi_{D_{orig,h}}$), resembling the natural model. Next-hop selection is realized using a probabilistic method. A message heading towards destination D_{dest}, arriving from node h at node i is sent to node j, $j \in N_{D_{dest}}$ (i.e. j is in the next-hop table for the destination D_{dest}), with the probability

$$p_{D_{dest,j}} = \frac{\psi_{D_{dest,j}}}{\sum_{k \in N_{D_{dest}}} \psi_{D_{dest,k}}} \tag{4}$$

After forwarding a message, the fields *Last hop*, *Hops*, and the *History* are updated. Decay of pheromone is regularly done as described in equation (1).

2.2.2 Data Migration

After successful discovery of the required SES_{id} segment, the data are sent using the routing mechanism described above. When data arrives, the hosting RSoC sends back a confirmation to the requesting RSoC about receiving data. The requester waits for a confirmation for a certain amount of time. When the timeout expires, the requesting RSoC repeats the segment discovery process in order to find another SES_{id} provider.

2.3 Optimization

As seen in Figure 2, each SES other than the last SES can migrate its processed output data to either one of two subsequent SESs implementations. This creates the challenge of finding the best configuration to execute a service in terms of communication time and execution time. It is not always true that choosing a close SES is the best solution. This becomes clear if we look at the complete path. For example, in Figure 2, it may seem a better choice to chose $SES_{2,h}$ as it is closer to $SES_{1,h}$, but this is not the case looking at the complete path. Other considerations such as network congestion and latency should be taken into account. Moreover, this stage may also involve identifying errors that may occur due to a bad SES implementation, the RSoC's possible lack of power, or RSoC failure. The optimization stage can take place in conjunction with the second stage. This can be done by sending exploration messages in ideal time to find other possible SESs paths.

2.4 Self-organization

The self-organization stage is important to organize and ensure the availability of all the SESs belonging to one service. In this stage, the SESs are to be moved and reallocated in RSoCs where they are relatively close to each other. Moreover, these RSoCs are also close to where the service is mostly acquired. The self-organization stage can take place after several runs of the system. This is to ensure the correctness and the validity of the statistical data acquired. Such data includes the frequency and the places where a service is most requested.

3. RELATED WORK

Approaches to discover distributed entities have been proposed in different contexts. To find resources or entities situated on different nodes in a network, the classical link-state protocol [4] maintains a global view of the entire network at each node. To keep this information up-to-date, each node periodically initiates a flood of the network. Naturally, this method does not scale well. To improve scalability, Perkins and Bhagwat [4] proposed DSDV, which is to a certain extent similar to the link-state approach but only maintains the information about the destination, the next hop towards it, the corresponding distance, and sequence number. Motivated by the fact that most of the data exchanged by DSDV is not needed (e.g. since a certain entity is of no interest) and changes in the topology cause a network flood, Perkins and Royer proposed AODV. Being similar to DSDV, it only discovers remote entities and repairs routes when needed, i.e. in an ad hoc manner. As DSDV and AODV discover only a single entity and path to such entity that matches a certain requirement (e.g. a certain node ID), both are not suited for inter-service communication on distributed RSoCs. Protocols that aim to enable this functionality, uniting multicast and multipath methods, such as ADMR [5], however fail to exhibit a behavior that adequately reflects the underlying network properties, e.g., the distance to, the quality of, or the quality of the path towards the remote entity. Several further approaches have been proposed that ignore the network topology and instead focus on the semantic level of service discovery: One of the most prominent examples is Sun Jini [6], which relies on a central service directory, where providers register and clients send queries to, which unfortunately represents a single point of failure. SLP [7] is in many ways similar to Jini, however, it supports multiple directories (i.e. directory agents).
Further benefits and related work for implementing OS in hardware can be found in our previous work [1].

4. CONCLUSION

The introduced work is interesting and may prove to be the seed for a new OS generation. However, each of the introduced challenges is in need of a more distant investigation. This also includes communication time minimization, and automated partitioning methods.

5. REFERENCES

[1] S. Samara, F. B. Tariq, T. Kerstan, and K. Stahl, "Applications adaptable execution path for operating system services on a distributed reconfigurable system on chip," in *ICESS '09*. IEEE Computer Society, 2009, pp. 461–466.

[2] P. Janacik, O. Kao, and U. Rerrer, "An approach combining routing and resource sharing in wireless ad hoc networks using swarm-intelligence," in *Proceedings of the 7th ACM/IEEE International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM 2004)*, 2004, poster session.

[3] P.-P. Grassé, "La reconstruction du nid et les coordinations interindividuelles chezbellicositermes natalensis etcubitermes sp. la théorie de la stigmergie: Essai d'interprétation du comportement des termites constructeurs," *Insectes Sociaux*, vol. 6, no. 1, pp. 41–80, March 1959.

[4] C. E. Perkins and P. Bhagwat, "Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers," vol. 24, no. 4. New York, NY, USA: ACM, 1994, pp. 234–244.

[5] J. G. Jetcheva and D. B. Johnson, "Adaptive demand-driven multicast routing in multi-hop wireless ad hoc networks," in *MobiHoc '01: Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*. New York, NY, USA: ACM, 2001, pp. 33–44.

[6] Jini.org, online, accessed July 9, 2009. [Online]. Available: $http://www.jini.org/wiki/Category:Introduction_{to}Jini$

[7] J. Veizades, E. Guttman, C. Perkins, and S. Kaplan, "Service location protocol." [Online]. Available: http://www.ietf.org/rfc/rfc2165.txt