Dynamic Partial Reconfiguration of the Ubichip for Implementing Adaptive Size Incremental Topologies

Héctor F. Satizábal, Andres Upegui

Abstract—The Ubichip is a reconfigurable digital circuit with special bio-inspired mechanisms that supports dynamic partial reconfigurability in a flexible and efficient way. This paper presents an adaptive size neural network model with incremental learning that exploits these capabilities by creating new neurons and connections whenever it is needed and by destroying them when they are not used during some time. This neural network, composed of a perception layer and an action layer, is validated on a robot simulator, where neurons are created under the presence of new perceptions. Furthermore, links between perceptions and actions are created, reinforced, and destroyed following a Hebbian approach. In this way, the neural controller creates a model of its specific environment, and learns how to behave in it. The neural controller is also able to adapt to a new environment by forgetting previously unused knowledge, freeing thus hardware resources. We present some results about the neural controller and how it manages to characterize some specific environments by exploiting the dynamic hardware topology support offered by the ubiquip.

I. INTRODUCTION

Nowadays technological trends are allowing higher system integration at lower costs. Current semiconductor integration technologies allow to include billions of transistors in a single chip, reducing in a dramatic way the cost per transistor. This fact has allowed the appearance of pervasive systems, myriads of small distributed embedded systems equipped with sensing, actuating, and/or communication capabilities, which are constantly interacting with the environment and with human users.

These systems are typically confronted to changing environments or to users with different preferences. Moreover, given the high number of devices, it becomes very difficult to manually upgrade, tune, or customize them, for fitting the requirements for a specific user or for specific environmental conditions. Adaptation becomes thus a very desirable feature for such type of systems, in order to allow them to meet the specific requirements for each of the possible scenarios.

Different levels of adaptation can be identified, ranging from simple parameter tuning to complicated topological modifications which may determine the internal system structure. Moreover, when we consider the case of constantly changing environments, the system should continue adapting in an incremental manner. This dynamic behaviour is performed naturally by living beings at different levels. For instance, biological nervous systems are in continuous adaptation during the developmental and learning processes. Hence during these periods, neurons are constantly replicating, differentiating, migrating, creating synapses, strengthening them, or destroying them according to the genetic plan of the individual, and its interaction with its environment.

Biological nervous systems, and specially biological neurons and its interconnections, have been used as inspiration to mimetically endow artificial systems with properties belonging to living beings like learning and adaptation. Artificial neural networks (ANN) constitute a clear example of this approach. Since the introduction of ANN as an alternative data processing tool, a large number of topologies and implementations of learning algorithms have been created [1]. There are networks which learn in a supervised or unsupervised way; networks having one, two, or more layers of neurons; networks performing classification, or regression tasks; fixed size, and adaptive size networks, etc.

Nowadays, ANN are powerful tools for the implementation of mobile robot controllers. Nonetheless, the size of a network is always hard to determine, and it is even harder when the environment changes during time. Several neural models featuring incremental adaptive topologies have been proposed in order to automate the network construction, and to enable the model to solve incremental tasks [2]. Some examples of such networks are ART [3], GAR [4], FAST [5], and GNG [6]. In a more general framework, these techniques offer the possibility of adapting the structure of the problem solver to the complexity of the problem at hand. This same principle can also be exploited by the field of dynamically reconfigurable hardware systems in order to provide simultaneously high performance, adaptability, and fault tolerance. However, its application on currently available reconfigurable devices is not straightforward. Several approaches have been proposed for exploiting the flexibility of reconfigurable circuits in order to implement adaptable topologies, each of them exhibiting a set of limitations [7]. Bearing in mind these limitations, in [8] they have proposed the ubiquip architecture.

The Ubichip is a reconfigurable digital circuit with bio-inspired mechanisms that allows a flexible and efficient implementation of hardware systems featuring dynamic topologies. It supports the implementation of self-replicating systems, dynamic creation and destruction of interconnections, and a very flexible dynamic partial reconfigurability. It has been used for modelling synaptogenic neural networks [9],
In this paper, we present an application of the reconfigurability capabilities of the Ubichip in the automatic dynamic partial configuration of a robot controller. The proposed controller is a two-layered adaptive size neural network which adapts its topology according to the environment and behaviour of the robot. One of the layers of the network is implemented on the chip, and the dynamic partial reconfigurability of the device is used in order to create or destroy neurons when needed. For validating our approach we use a robotic simulation platform called Enki, programmed using the ASEBA event-based architecture [12], and running on a virtual environment inspired from Aseba-Challenge [13]. We defined a set of arenas where our neural controllers will be trained, and we analysed the resulting networks obtained in function of the specific environmental conditions.

The rest of the article is organized as follows. Section II introduces the ubichip architecture, and the ubichip reconfigurability mechanisms exploited by our work. Section III introduces the adaptive size incremental neural network to be used by our approach, and section IV describes its implementation on the ubichip. Then, sections V and VI describe the different experimental scenarios setup for the robotics simulator and discuss their respective results. Finally, section VII concludes and discusses future work on this line.

II. THE UBICHIPI

The ubichip is a custom reconfigurable electronic device capable of implementing bio-inspired mechanisms such as growth, learning, and evolution [8]. These bio-inspired mechanisms are possible thanks to reconfigurability mechanisms like dynamic routing, distributed self-reconfiguration, and a simplified connectivity. Moreover, the ubichip offers a fast parallel configuration interface that allows a processor to readback and to configure the circuit. The reconfiguration can be executed in a dynamic and partial way supporting a fine grain reconfigurability.

A. Ubichip Architecture

The ubichip is mainly composed of three reconfigurable interconnected layers. Units from each of the three different layers are grouped for forming a macrocell. A macrocell contains thus four ubicells connected to a routing unit and a self-reconfiguration unit as described in [8].

The first layer is an array of ubicells, the reconfigurable logic elements used for computation purposes. A ubicell is composed of four 4-input look-up tables (LUT) and four flip-flops (DFFs). These ubicells can be configured in different modes like counter, FSM, shift-register, 64-bit LFSR, adder, subtracter, etc. An ubicell can also implement a simple 4-bit processing element being part of a SIMD multiprocessing platform, and $n$ ubicells can be merged to create a $4n$-bit processor.

The second layer is made of self-reconfiguration units that allow part of the circuit to self-replicate somewhere else on the chip, without any external intervention [14]. The third layer contains dynamic routing units connected to their eight neighbours which permit the ubicells to dynamically connect to any part of the circuit. In this layer, routing paths can be created at run-time, destroyed, and get connected to different units, in a completely autonomous manner [9]. These last two layers are not exploited by the work presented in this paper.

B. Reconfigurability

One of the goals of the ubichip, is to offer a high flexibility and efficiency for implementing dynamic partial reconfigurable systems. This flexibility is provided thanks to a fast configuration access and a fine granularity reconfiguration.

The configuration of the ubichip can be done through a parallel 16-bit configuration bus, or through a serial configuration interface composed of two input lines and one output line. The parallel configuration port has been designed to be controlled from an XScale PXA270 processor through its memory access bus using the Variable Latency Input Output (VLIo) interface, as described in [15]. In this way, one can access any configuration register of the Ubichip by addressing a memory position from the PXA270. The configuration granularity allows to write and read 16-bits words from the configuration bit-string with a single memory access from the processor, and without any configuration header because the configuration registers are considered as a memory extension from the processor side. This fine granularity offers a very high flexibility compared to current state-of-the-art commercial devices. For instance, the minimal addressable configuration bit-string section on Virtex-5 devices from Xilinx is a frame, defined with 41 words of 32 bits [16]. Moreover, any configuration access requires an additional configuration overhead of 272 words. This frame-oriented addressing, and the associated header makes small accesses very time-consuming, respective to our configuration port.

The Ubichip also provides a set of special configuration control registers that allow to run the circuit in step-by-step or in free-run mode. These modes allow to control the circuit execution from the PXA270 in order to stop the circuit whenever it is needed. In most cases, stopping the circuit is important when partially modifying the configuration bit-string of a circuit. It guarantees that the configured circuit will have a stable state before and after the partial reconfiguration.

The full configuration of an ubicell is done in 8 memory access from the PXA270 (around 0.6$\mu$s) and a complete Macrocell is configured in 37 access (around 2.77$\mu$s). The access time depends on the Ubichip operation clock frequency. This is the access time measured on an FPGA prototype with an operation clock frequency of 40MHz. However, the final circuit is expected to run at 60MHz. The partial configuration of a Virtex-5 functional unit like a CLB would require to reconfigure 36 frames, what implies to configure 56064 bits including the header. The time required for this reconfiguration would be 70$\mu$s by using the fastest configuration interface (SelectMap 800Mb/s) [16]. Even if it is not possible to establish a straightforward comparison
between our Macrocell and a CLB, we have anyway a configuration addressing that allows to reconfigure our basic building block around 25 times faster than the Virtex-5 block. In the case that the targeted reconfiguration is not a building block but a single configuration bit, the time for configuring it would be $0.075 \mu s$ on the ubichip and $12.52 \mu s$ for the Virtex-5. The fact of having a configuration interface being 167 times faster than current commercial state-of-the-art devices, makes the ubichip a promising solution for systems requiring frequent fine grained partial reconfigurations.

III. INCREMENTAL PERCEPTION-ACTION MAPPING

Two factors guide the adaptation of living organisms, their genetic material and the interaction with their environment. The former is mainly present in developmental processes, where individuals grow and incrementally become capable of performing more complex tasks. The latter is mainly present in the subsequent processes of learning, when individuals adapt their behaviours in order to meet more specific requirements of the changing environment by acquiring new knowledge and dexterities. As far as learning is concerned, from small multicellular organisms to the more recent species of mammals, all these organisms performing complex behavioural adaptations share a key component: the existence of a nervous system with specialized cells called neurons. These neurons, which connect the sensory cells of the organism to its motor counterparts, increase the possibilities of regulating the activity of motor cells. Hence, having a nervous system allows the generation of more complex motor responses than in the case of organisms lacking it. In such a case, the organism performs the fundamental behaviours that are necessary for survival of the individual (i.e. ingestive, digestive, and reproductive) by means of simpler motor responses called taxis\(^1\), but without any associative learning of tasks [17].

Learning and biological nervous systems have been used as inspiration in the creation of different control strategies, and in particular, in robot controllers, where several approaches have been considered [18]. Artificial neural networks can be used as robot controllers in the form of Correlational procedures which are often used to represent a given space in a compact and topology-preserving manner, by performing feature discovering or clustering in the input and/or output spaces. Conversely, error-minimization procedures do not perform feature discovering, but use complete target information in order to build input/output mappings. Reinforcement-based procedures lie between the two last approaches, and are trial-and-error procedures permitting the exploration of different input/output relations by building mappings that maximizes a reward signal.

Our model is based on the correlational approach mentioned above, and takes inspiration from the layered neural structures in biological nervous systems. Neurons in simple nervous systems are organized in two layers\(^2\); a sensory

layer with sensory neurons which receive stimuli from the environment, and a motor layer of motoneurons which innervate effector cells [17]. The complex behaviour of simple animals is thus due to the interconnection of these two layers. This organization is present even in mammals, where the sensory and motor cortex are juxtaposed and linked by connections [19].

The model of neural controller we propose also has two layers of neurons. The first layer, called the perception layer, performs feature detection on the input space by clustering the signals coming from the sensors of the robot (two proximity infrared sensors in this case). In the same way, the second layer, we called the action layer, performs feature detection on the output space by processing analogously the speed of the motors. Interconnections between these two layers are created by means of a Hebbian approach [20] in order to link perceptions to actions. In this way, input signals will activate some neurons in the perception layer, that will activate some output neurons in the action layer driven by the interconnection matrix, in order to set the actual system outputs. In our model, one neuron in the perception layer can activate several neurons in the action layer, and one neuron in the action layer can be activated by different neurons in the perception layer. These are properties known as divergence and convergence in biological nervous systems [17].

In order to train the whole network (i.e. perception and motor layer, and their connections), an external teacher has to drive the robot within its environment. During this phase, the neural controller characterizes the input and output spaces, and creates links between them according to the navigation behaviour shown by the external driver\(^3\).

Other examples of neural controllers are the one presented by Perez-Urrie in [21], and the one presented by Zrehen in [22]. These neural controllers, even if they use a similar dynamic characterization of the sensory space, do not perform any dynamic exploration of the output space (i.e. motor speeds). Moreover, in both approaches a reinforcement-based procedure is used in order to create a mapping between the input and output fields.

A. Adaptive Size Incremental Topology

The perception and motor layers contain representations of the observed world, and of the responses of the robot actuators respectively. This representation is stored in the form of a growing layer of neurons we called Adaptive Size Incremental Topology (ASIT), that increases its size when new observations are experienced. The operation of the ASIT is sketched as follows.

The first neuron in the ASIT is created at the position of the first observation presented to it. Afterwards, when a new observation is presented to the network, every existing neuron computes whether it already knew the observation or not. To do so, every neuron calculates the distance to the new observation, and compares it against a claim threshold.

\(^1\)E.g., phototaxis, chemotaxis

\(^2\)Nervous systems of simple animals like hydra, or in general, all the phylum Cnidaria [17].

\(^3\)A human driver was used in this case, but it could be a software agent performing a specific navigation algorithm (e.g. a Braitenberg vehicle)
If any neuron claims to know the observation, a new neuron is created at this position. Otherwise, if the new observation is located within the claim area of a neuron, the distance from the claiming neuron to the new observation is compared against a move threshold and, if the new point is found to be inside of the move region, the position of the claiming neuron is updated to the new observation with a certain probability.

Moreover, ASIT is also able to forget perceptions. Each neuron has an age value associated, which is reset every time the neuron is activated (claim). When a neuron’s age arrives to a certain maximum life expectancy threshold, the neuron dies forgetting not only the information stored in the neuron, but also every possible connection from it.

Other networks able to carry out incremental topology adaptations are Growing Neural Gas [6], the FAST architecture [5], the GAR network [4], and ART [3]. These artificial neural network topologies learn incrementally by placing prototype neurons according to constant parameters and local measures of error. These two properties make them able to be used in solving incremental tasks [2] like the interactive creation of navigation controllers in mobile robotics.

### B. Associating Perceptions to Actions

Connections between the perception and action layers are created, reinforced, weakened, and destroyed, following a Hebbian learning approach. During training, whenever there is a coincidence in the firing of a neuron in the perception layer and another neuron in the action layer a connection between these two neurons is created, and an initial weight is associated to it. If the connection already exists, its associated weight is strengthened, while the rest of connections arriving to the firing action neuron are weakened. Additionally, connections that are not used are constantly weakened during time, allowing the network to prune unused connections in order to avoid learning noisy observations.

### C. Operation of the Model as a Mobile Robot Controller

The presented model do operate in two modes, a training and a running mode. The training mode (as it has been mentioned in the beginning of this section) performs feature detection on input and output spaces (sensors and motors) in order to characterize the environment and the behaviour of the robot. This process is completely unsupervised for both layers. Nonetheless, an external agent has to drive the robot in order to show the desired behaviour of the robot to the system. This behaviour is saved in the form of connections between the perception and the action layer, as it was mentioned in subsection III-B. Thus, the learning process of the complete system is supervised, even if both layers perform in an unsupervised way, and the connections are created by following a Hebbian rule. The training mode is sketched in figure 1.

The running mode enables the controller to actually control the behaviour of the robot according to the information gathered during previous training phases, and disables the modifications in the structure of the network. During this mode, the external driver has no effect on the behaviour of the robot, and the speed of the motors depends on the outputs of the controller, which are computed as is explained in the following three steps:

- Initially, a new observation of the current environment is taken from the sensors. This new input is fed to the perception layer in order to get the activation of the existent neurons (features).
- Next, if none of the neurons do claim, it means that the controller do not know the current environmental conditions. In this case, the controller makes the robot perform an “innate” reflex behaviour, which in our case was fixed to simply going back. Conversely, if one or more perception neurons do claim, the action neurons connected to the perception claiming neurons are activated.
- If any of the action neurons get activated, then the output of the system is computed by doing a weighted sum of the information (motor speeds in this case) of every claiming unit on the action layer. The weight of each action neuron is the weight of the connection coming from the perception layer which has activated it.

The running mode is sketched in figure 2.

### IV. THE ASIT ON THE UBICHRIP

An ASIT representing the perception layer has been implemented on the ubichip architecture. The action layer and interconnections between layers have been implemented in software in the form of a plugin that interacts with the UbiManager (The tool used for managing the ubichip configuration) [23].

The architecture of the implemented neuron is depicted in figure 3. Each neuron stores its current position on a Point Register \(p_i \text{reg} \), where each register corresponds to a dimension on the problem to solve (the number of sensors, in our case). The neuron has also 4 inputs coming from sensors \(s_i \text{input} \), which are used for computing the
Manhattan distance between inputs and current stored values. This distance is then compared to two constant values stored in two more registers. If the distance is smaller than a Claim Threshold value, the Claim output is set to ‘1’. That means that the neuron claims that the input vector is within its knowledge range, and it will activate in order to trigger its corresponding output neurons. The comparison with the Move Threshold value is used for determining whether the neuron may update its current weight or not. Moreover, this decision is not deterministic but stochastic, when the distance is smaller than the threshold a pseudo-random bit generator with a certain probability of generating ‘1’s will decide whether Point Registers are updated or not.

This architecture was implemented on the ubichip using a bus resolution of 4 bits. A neuron implementation requires an array of $5 \times 8$ ubicells, and its implementation on the UbiManager tool is depicted in figure 4. In this figure it can be seen an array of $20 \times 10$ ubicells, that are initially configured with a neuron in the left section of the circuit. Inputs from sensors are introduced to the four registers highlighted in the bottom right part of the figure, which are propagated to every existing neuron. In this example, there are still enough available reconfigurable units for replicating 2 more neurons, for a maximum total of 3 neurons. It must be noted that the implementations presented in this work use a simulated description of the ubichip because the real circuit is currently being fabricated. However the final chip will dispose of an array of $20 \times 20$ ubicells, and it will be possible to have several chips on the same board.

The growing and pruning mechanisms of the perception layer contained on the ubichip are controlled by an application running on a plugin, which is also establishing the interface with the robot simulator. For dealing with the process of creating and destroying neurons, in each algorithm iteration the plugin performs the following 6 steps: (1) getting robot’s sensor data from the robot simulator, (2) introducing sensor data to the ubichip, (3) checking neuron claims on the ubichip, (4) creating new neurons if there no claims, (5) killing old unused neurons, and (6) killing redundant neurons. The plugin contains also the action layer, which follows the same sequence that the perception layer, with two main differences: (1) data does not comes from the robot simulator but directly from the teacher, and (2) the ubichip is not involved in the operation of this layer.

In order to validate our algorithmic approach, we configured a set of experiments were our robot was confronted to different scenarios requiring different perception-action mappings. In this way, the network should learn how to map the specific situations of a given setup to a set of actions. The observations of two different setups may be different and may require a different number of neurons, and of course a different perception-action mapping. The information fed into the perception layer came from two of the infrared proximity sensors of the robot.

We considered a set of different arenas where the robots evolved. All of them are circular arenas as depicted in figure 5 where, during learning, the robot is trained for navigating through them in clockwise and anticlockwise senses. The robot has been trained on arenas where the path has different widths, and the resulting obtained networks are analyzed.

![Fig. 4. Ubichip implementation of a 4-input neuron. The neuron is implemented on 8x5 ubicells at the left. The remaining empty ubicells are available for further neural replications.](image1)

![Fig. 5. Uniform path width arena.](image2)
The network learning includes the ability of forgetting old unused knowledge. This is a very important aspect since this allow to reuse available resources for other neurons, or other computing tasks. We used the arena depicted in figure 6 in order to validate its functionality. Two circular arenas with different path widths are used. The robot learns how to navigate in one arena, and then it goes to the second one. A part of its previous knowledge is reusable for evolving in this new environment. During some time, the neural controller knows how to navigate in both environments, but after some time of not being confronted to the former one, it forgets the unnecessary perceptions, by destroying the old neurons.

VI. RESULTS

This section shows some qualitative results of the operation of the proposed neural controller on different environments. Even if the different arenas are just slightly different, it is enough from the point of view of the perception of the robot to generate completely different input mappings.

A. Different Perception Mappings for Different Arenas

The circuit shown in figure 5 has a radius of 30cm, and was configured to have different path widths, from 9 cm to 12 cm. Figure 7 shows the resulting network configuration when the robot is driven through a path with a width of 9 cm. In this first example the robot has to pass very close to the walls when going through the circuit, making the sensors to produce high response values. This fact is captured by the perception layer which successfully captures the distribution of the data obtained from the sensors.

The resulting network structures after driving the robot through different path widths is shown from figure 8 to figure 10. Figure 8 shows that when having more room, the sensors of the robot produce values with a different distribution, and how this behaviour is captured by the perception layer.

The experiment shown in figure 10 was done with an arena where the robot is unable to see both walls at the same time. This condition is reproduced by the distribution of neurons in the perception layer, which in this case has neurons near to the axes.

The experiments shown previously were done in uniform arenas having a constant path width. Figure 11 shows the resulting network structure after driving the robot through a circuit having a non-uniform path width. In this case, the external wall describes an ellipse which produces a non-uniform path width ranging from 10 cm to 12 cm. Hence, there are places where the robot has to pass close to both walls, generating observations far from the origin; and other regions where the robot has more room to move, generating observations close to the axes.

It is worth to say that, after training, the neural controller drives the robot without hitting the walls in all the configurations shown above. In every case, the training procedure
produced neural topologies describing successfully the input and output spaces of the robot. Moreover, the connections between the perception and the action layer were good enough for allowing the robot to evolve in its reduced environment in an autonomous way.

B. Forgetting old Perception-Action Mappings

The experiments shown in the previous subsection were independent trials where empty structures were used as initial networks. In this section, it is shown a sequential experiment aiming to show, not only how the network is able to increase its size with new perceptions, but also how the network is able to forget unused knowledge in order to free hardware resources. The circuit used to do this test is shown in figure 6. The ring on the left side has a path width of 12 cm, and the ring on the right side has a path width of 9 cm.

Figure 12 shows the structure of the network after driving the robot through the ring at the left side of the circuit (12 cm). The network structure is similar to the one shown in figure 10, since the path width is the same as in that case. Neurons in the perception layer are placed close to the axis since the sensors never detect both walls at the same time.

Figure 13 shows the network structure after driving the robot through the ring located at the right side of the circuit (9 cm). The network structure was not initialized between this step and the previous one. Hence, besides the neurons created due to perceptions of close walls, there are neurons in the perception layer which were created by old perceptions from the leftmost ring. These old neurons remain in the layer even if they are not further activated, and thus, the model keeps the previous obtained knowledge.

The situation changes when time passes and previous knowledge becomes irrelevant. After some time of activating only neurons belonging to perceptions of the rightmost ring, old neurons encoding features from the leftmost ring begins to die due to the large time without being activated. This situation is shown in figure 14.

Finally, after passing more time driving the robot through the ring located at the right side of the circuit, all the neurons generated by perceptions from the leftmost ring die, freeing hardware resources, and enabling a better representation of the relevant environment.

In our model, parameter \textit{maximum life expectancy} controls the forgetting behaviour of the network. higher values of this parameter produces network structures which memorizes perceptions after longer periods of inactivity.

VII. CONCLUSIONS

In this paper we have introduced the reconfigurability features that make the ubichip a promising solution for the implementation of adaptive pervasive systems, and we have presented the case of an adaptive size neural controller that adapts its topology in order to characterize its perceptions.
from the environment. This adaptive neural controller has been validated in different arenas where the observations for the robot changed during the robot life-time. The robot neural controller was able to learn how to behave in an initial environment, it was then able to learn how to behave in a different environment with different observations, and finally, it was able to forget its initial behaviour after a certain time of not using it any more. We have shown how this approach was able to benefit from the dynamic partial hardware reconfigurability offered by the ubichip, where the fact of forgetting allowed to reuse hardware resources for including new behaviours.

In the current implementation, the action layer and the interconnection among perception and action layer is done in software. However, in a further work it is envisioned to implement the whole network in the ubichip by using the dynamic routing for creating the links between layers.

Moreover, in the presented work the adaptation was performed in the form of a supervised learning approach were the teacher was a human. In a more general framework, we can consider the case where the teacher is not a human but another device, that may have a different set of sensors (different number, different type, and/or different calibration). An initial device (the teacher) that knows how to solve the task (tuned by a human) can teach other devices (learners) how to perform a task just by indicating them the desired output in a given situation. After some time, learners will be able to map their perceptions to actions as the teacher showed them, for further becoming teachers. In this way, one can foresee the possibility of training a myriad of adaptive pervasive systems for performing a new task by introducing a single teacher.

ACKNOWLEDGMENTS

We thank Philippe Retornaz, Stphane Magnenat, and Francesco Mondada for their support in the use of Aseba and Enki, and Yann Thoma for his support on the use of UbiManager.

This project is funded by the Future and Emerging Technologies programme IST-STREP of the European Community, under grant IST-034632 (PERPLEXUS). The information provided is the sole responsibility of the authors and does not reflect the Community’s opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

REFERENCES