EPiCS: Engineering Proprioception in Computing Systems

Tobias Becker, Imperial College London, UK
Andreas Agne, University of Paderborn, Germany
Peter R. Lewis, Rami Bahsoon and Funmilade Faniyi, CERCIA, University of Birmingham, UK
Lukas Esterle, Klagenfurt University, Austria
Ariane Keller, ETH Zürich, Switzerland
Arjun Chandra and Alexander Refsum Jensenius, fourMs, University of Oslo, Norway
Stephan C. Stilkerich, EADS Innovation Works, Germany

***Abstract*—Modern compute systems continue to evolve towards increasingly complex, heterogeneous and distributed architectures. At the same time, functionality and performance are no longer the only aspects when developing applications for such systems, and additional concerns such as flexibility, power efficiency, resource usage, reliability and cost are becoming increasingly important. This does not only raise the question of how to efficiently develop applications for such systems, but also how to cope with dynamic changes in the application behaviour or the system environment.**

The EPiCS Project aims to address these aspects through exploring self-awareness and self-expression. Self-awareness allows systems and applications to gather and maintain information about their current state and environment, and reason about their behaviour. Self-expression enables systems to adapt their behaviour autonomously to changing conditions. Innovations in EPiCS are based on systematic integration of research in concepts and foundations, customisable hardware/software platforms and operating systems, and self-aware networking and middleware infrastructure. The developed technologies are validated in three application domains: computational finance, distributed smart cameras and interactive mobile media systems.

I. Introduction

Designing and operating today's computing systems is becoming increasingly challenging for a multitude of reasons. Compute nodes evolve towards parallel and heterogeneous architectures to deliver continued performance gains. Distributed systems grow in size and complexity, and the network topology and the available resources of the system can vary during run time. Systems must be able to cope with these increasing levels of dynamic behaviour. Finally, future application domains have divergent requirements with respect to functionality and flexibility, performance, power efficiency and reliability, and these requirements may also change at run time. Current static design paradigms do not scale with requirements of developing applications for increasingly parallel, heterogeneous and distributed systems. Furthermore, they are neither efficient in supporting the development of increasingly dynamic systems and applications, nor are they capable of managing changing requirements at run time. Novel design and operating principles are necessary to address these challenges.

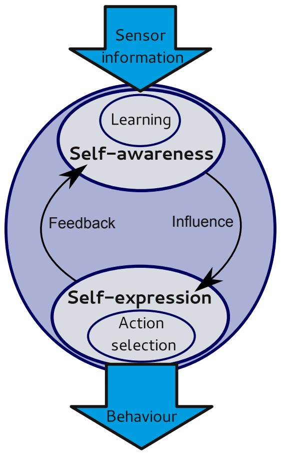

Fig. 1. Self-awareness and self-expression in a compute node.

Self-awareness is an emerging field of research in computing that studies a novel class of systems that can reason about their execution and adapt their behaviour if necessary. The EPiCS project aims at laying the foundation for engineering such self-aware and self-expressive computing systems. Self-awareness is enabled by maintaining information about the current state, observing oneself and the environment, and reasoning about the current behaviour. This knowledge is utilised for self-expression: the ability of a system to adapt its behaviour to changing conditions and environments. The general conceptual process of self-awareness and self-expression is illustrated in figure 1.

In the EPiCS project, we denote the basic ability to collect and maintain information about state and progress as proprioception, referring to psychology where proprioception (from Latin proprius, meaning ”one's own”, and perception) is defined as the sense of the relative position of neighbouring parts of the body. Proprioception is the enabler for building awareness and realising advanced autonomous behaviour. Proprioception, self-awareness, and self-expression are concepts mainly known from psychology, philosophy and medicine.

EPiCS aims to successfully transfer of these concepts to computing and networking domains, enabling powerful and versatile heterogeneous and distributed future systems and applications. This approach has also been followed for other nature-inspired computing paradigms, such as evolutionary computing and swarm intelligence, which despite their success often lack the assurances and guarantees required for the construction of technological systems. Hence, EPiCS also includes a thorough analysis of the limits of the approach and studies its suitability for different application domains.

The EPiCS Project explores self-awareness and self-expression through research in the following areas:

- General concepts and foundations for self-awareness and self-expression in technical systems.
- Requirements and validation.
- Hardware/software platform and operating system for self-awareness.
- Self-aware networking and middleware infrastructure.
- Three technology demonstrators that illustrate the developed technological innovations.

The EPiCS consortium consists of 8 institutions: University of Paderborn (Germany), Imperial College London (UK), University of Oslo (Norway), Klagenfurt University (Austria), University of Birmingham (UK), EADS Innovation Works (Germany), ETH Zürich (Switzerland), and AIT Austrian Institute of Technology (Austria).

II. Working Areas

A. Concepts and Foundations for Self-Awareness and Self-Expression

In order to lay the foundations for novel systems that exhibit self-awareness and self-expression, it is important to study the fundamental concepts which support both these systems themselves and the principled development of them. We therefore conduct foundational research firstly into what self-awareness and self-expression concepts might mean for computing systems, including their benefits and limitations, and secondly into novel techniques to implement self-awareness and self-expression in systems characterised by decentralisation, heterogeneity, dynamism and self-organisation. A key hypothesis investigated in the EPiCS project is as follows:

1) That systems which are *aware* of their own state, behaviour and performance can manage trade-offs between goals at run-time, and
2) That this enables them to better meet their requirements in uncertain and dynamic environments.

Our approach is to design systems as collectives of self-aware, self-expressive nodes, which learn and interact in order to self-adapt and self-organise. Online learning is used to enable run-time adaptation, reducing design-time overhead. Self-aware and self-expressive nodes engage in online algorithm selection, to better meet their own goals. Furthermore, node interaction mechanisms drive global behaviour, leading to a collective behaviour which provides the functionality of the system.

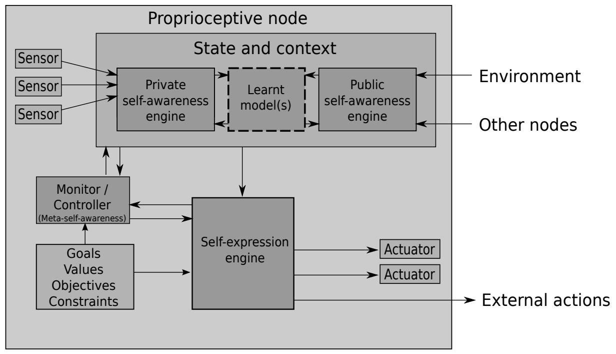

Fig. 2. Reference architectural framework for a proprioceptive node.

In this context, we are interested in two key conceptual questions. Firstly, are there benefits associated with increased levels of self-awareness in computing systems, and if so, under what conditions? Secondly, if this is the case, then how can we design self-aware systems?

In order to understand the potential benefits of self-awareness and self-expression for computing and engineering, we have conducted a survey both of those concepts as understood in psychology and also of previous efforts to apply them to computing [9]. A key finding of our survey is that the term self-awareness has been used in a variety of ways within computer science and engineering literature. Often however, it refers to quite disparate ideas, for example to highlight specific self-monitoring capabilities of a system, to indicate an awareness by the system of the user or context, or that a component has a conceptual knowledge of the wider system of which it is part. The general concept of self-aware computing covers but is not limited to all of these cases.

Therefore, our ongoing work is concerned with the translation of psychological concepts into a computing context, thereby presenting a framework for self-aware computing. The framework is based on three key concepts [9]: that computing systems can possess public and private self-awareness, that the extent of a system's self-awareness capabilities can be characterised by levels of self-awareness which describe increasing capabilities, and that self-awareness can be an emergent phenomenon in collective systems. Public and private self-awareness are discussed in a computational context in [9], but can be summarised as follows:

1) **Private self-awareness**: A node's knowledge or perception of internal phenomena (e.g. internal state).
2) **Public self-awareness**: A node's knowledge or perception of external phenomena (e.g. environment or interaction with other nodes).

In order to structure the requirements for, and more widely aid the design of self-aware, self-expressive nodes, a reference architectural framework for proprioceptive nodes was developed, and is shown in figure 2. This reference architectural framework builds on standard agent architectures [15], by clearly identifying conceptual components responsible for self-awareness and self-expression. The architecture deals with the key concept of public and private self-awareness, by specifying conceptual components for building knowledge from each source of information.

One key challenge in realising self-awareness and self-expression is the development and application of online learning schemes, suitable for use in dynamic self-organising systems. Online learning is applied primarily to achieve two purposes within a self-aware and self-expressive system: at the adaptation level and at the meta level. Briefly, online learning is applied in these contexts as follows:

At the **adaptation level**, a self-aware node should be able to learn to recognise and associate meaning with characteristics present in the environment and interactions with it. Additionally, to be self-expressive, it should be able to learn high performing behavioural strategies. We find that a combination of social, economic and nature-inspired techniques lend themselves particularly well to enabling online adaptation in the presence of changing and uncertain scenarios (e.g [4], [10]).

At the **meta level**, a self-aware node recognises that a trade-off exists between multiple objectives specified in the node's requirements, and that optimising this trade-off will require different approaches in different situations [11]. A particular approach used at the adaptation level will give rise to a particular outcome in terms of this trade-off. The ability to optimise the way in which the node adapts to unforeseen scenarios and user preferences will require the ability to select between adaptation approaches at run time.

However, in uncertain and dynamic environments, a trade-off exists between the resources spent actually performing the task at hand, given current knowledge (or *awareness*), and learning to better perform the task specified by the requirements. A self-aware node should be able to intelligently weigh up the cost and benefits of exploring new strategies and approaches, against an expectation of current performance. Additionally, the learning process itself can consume resources, and this should also be factored in. In the EPiCS project, we study both existing and novel online learning approaches to achieve capabilities at both the adaptation and meta levels. Techniques have been studied in a novel abstract problem, the *relevant neighbourhood selection problem* [11], which has high relevance to the EPiCS application demonstrators.

A key finding of this study was that no single strategy outperformed the others consistently across a range of scenarios. Indeed, scenarios could easily be found where even the most naive of strategies could outperform apparent state-of-the-art techniques. This highlighted the need for meta-self-aware strategy selection (the idea that a node is aware of and can reason about its own self-awareness), which we demonstrated [11] using the simple *epsilon-greedy* strategy. Here, meta-self-awareness describes the ability to observe the node's own performance, and to switch between a set of strategies at run-time. Though the meta-self-aware strategy performed more consistently well than other base strategies, our results strongly suggest that the benefits associated with meta-self-awareness will be heavily dependent on the scope of the adaptation being considered. In this case, when making claims about proprioceptive systems, it appears crucial to include a description of the scope of expected adaptation in any claim.

Finally, since proprioceptive systems are likely to exhibit highly distributed and decentralised decision making, in collectives of self-aware and self-expressive nodes with only local knowledge, our work in EPiCS also considers interactions between such nodes. Due to these characteristics, social and economic-inspired mechanisms are highly applicable as methods for managing such interactions. We have demonstrated the application of such techniques to object tracking handover in distributed smart camera networks [4], to facilitate conflict resolution in multi-user interactive mobile musical systems [3], as well as in service provisioning in cloud computing [6].

B. Requirements and Validation

EPiCS defines a structured development process based on Goal-Oriented Requirements Engineering (GORE) as well as a systematic validation strategy to provide a feasible way of engineering proprioceptive compute systems. We argue that Goal-Oriented Requirements Engineering is a promising approach for leveraging engineering requirements and capturing their dynamics for proprioceptive systems. Goals are prescriptive statements of intent whose satisfaction requires the cooperation of different components (originally called agents) in software and its environment [18]. Goals range from high-level to fine-grained technical prescriptions that can be assigned as responsibilities to single components. Goals may capture functional or quality properties. A functional goal captures a desired set of scenarios, which need to be realised by the system. A quality goal captures behavioural requirements, which constrain the functional goals. GORE models are organised in structures that can represent refinement and abstraction, and they also support tracing of high-level goals to corresponding architectural elements. A goal refinement graph can capture relationships among goals using AND/OR refinement links. AND refinements relate to goals that are satisfied when all its subgoals are satisfied. OR refinements require at least one of the subgoals to be satisfied.

Our use of GORE is motivated by the need for capturing scenarios related to self-awareness and self-expression based on three demonstrators (see section III). This activity intertwines the initial requirements of each demonstrator, the knowledge of the application domain and the use of the reference architectural model (see section II-A) which acts as a general blueprint for proprioceptive systems. Self-aware scenarios and the corresponding goals are related to various levels of self-awareness, design qualities (e.g. security, availability, performance, etc.) and the associated utilities. The interaction between various self-aware goals, their trade-offs, their associated utilities and their satisfiability criteria are modelled using in the AND/OR refinements, allowing traceability to the reference architectural model. Another objective of the modelling and refinement process is to make goals as measurable as possible. Through refinement we link goals to specific architectural components and mechanisms where we can observe a measure behavioural impact. Treating goals

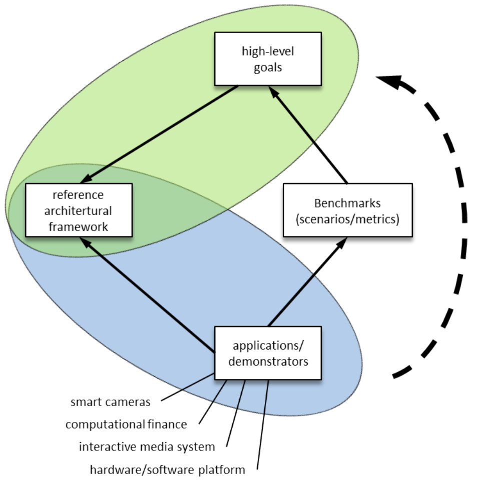
high-level goals
reference architertural framework
Benchmarks (scenarios/metrics)
applications/ demonstrators
smart cameras
computational finance
interactive media system
hardware/software platform

Fig. 3. Structured development and validation cycle.

which represent changes in functional nature is obviously less demanding than goals of non-functional nature, as the latter may cross-cut a set of architectural components.

Figure 3 depicts an extract of this process that starts with defining high-level goals that reflect the core features of the intended proprioceptive system. Based on these high-level goals and knowledge about the application domain, an initial set of requirements is derived. In the following, the goals and requirements are applied to the reference architecture framework, and the framework is refined accordingly. Next, the refined architecture is implemented to match the required functionality of the application. This proprioceptive system implementation is benchmarked using pre-defined, application-specific metrics. This allows us to verify that employing proprioception leads to actual improvements towards the high-level goals. Finally, all insights and lessons learned throughout this development process are fed back to first stage in order to refine the initial goals and the overall approach.

C. Self-Aware Hardware Software Platform

Based on the concept and foundations of self-awareness, we develop basic approaches and technologies for a self-aware compute platform. This compute platform leverages modern heterogeneous multi-core technology and uses a novel operating system that extends multi-threaded programming across the hardware/software boundary.

Autonomous compute node: A distinctive feature of EPiCS is to exploit run-time reconfigurable hardware such as FPGAs to enable self-expression within an autonomous compute node. Run-time reconfiguration allows us to move threads across the hardware/software boundary, a process we call vertical function migration. As an architectural basis, we built on the ReconOS [13] programming model and execution environment. ReconOS employs the multi-threaded programming model widely used in the software domain and extends it to the realm of reconfigurable hardware. This allows us to take hard-

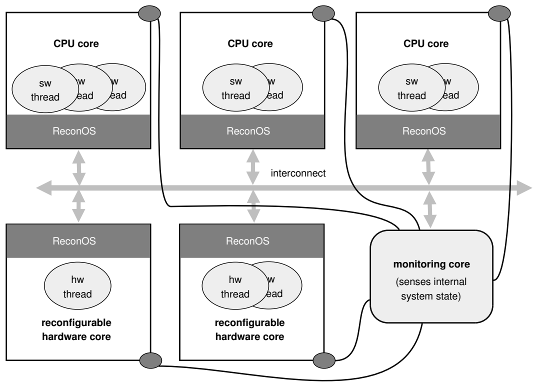

Fig. 4. Autonomous compute node with heterogeneous processors and monitoring core

ware accelerators and integrate them into the system as fully featured threads, that are able to share resources, communicate, and synchronize with other threads running under the common operating system. Our operating system builds on a current version of the Linux kernel, which allows us to use a wide spectrum of pre-existing libraries, applications, and operating system resources such as virtual memory management [1]. Figure 4 gives an overview of the autonomous compute node's architecture.

A key characteristic of a proprioceptive system is the system's awareness of its internal state (i.e. private self-awareness). The autonomous compute node achieves this by instantiating monitoring cores in the reconfigurable hardware that capture information about the system state, as for example thread-level performance and cache usage statistics. We also monitor the on-chip temperature distribution and use an adaptive temperature model [8] to predict future heat generation and heat flow. This enables us to avoid thermal hot-spots by proactive temperature-aware thread scheduling.

The self-expression capabilities of the autonomous compute node manifest themselves in the system's scheduling decisions. Our vision is to create a self-aware scheduler that is aware of its goals (e.g. thermal, performance, and fault tolerance constraints) and the system state as measured by the monitoring cores, uses adaptive models to predict future changes in the state, and bases its scheduling decisions (including the vertical migration of threads) on these predictions.

Methods for self-optimisation: EPiCS also investigate basic self-optimisation techniques to adapt specific functions of applications to the various resources of heterogeneous systems. These resources typically vary in their performance, efficiency and the way how they perform computation. If the nature of the computation changes, a heterogeneous system may respond by relocation the function to a different resource, adjusting the compute kernel on the current resource, or by adjusting the computation within the kernel. It is important that a self-optimisation process can take into account these various options and characteristics in order to improve the relevant metrics such as performance, power, resource usage, etc. The key principles for such a self-optimisation process are identifying application tuning parameters which are the basis for optimisation, developing instrumentation to collect run-time information, developing cost and performance models for re-parametrisation, and finally, creating mechanisms for executing re-parametrisation at run time.

We have identified hardware specialisation as one of the potential approaches for application tuning on reconfigurable hardware. Hardware specialisation refers to the concept of identifying constant or slowly changing inputs to a compute kernel [16]. The kernel can consequently be optimised for these fixed inputs, resulting in a specialised circuit with higher performance, lower area requirements and lower power consumption. If the inputs that are used for circuit specialisation change, the circuit will be reconfigured with a new specialised version. However, the overhead that is associated with replacing or reconfiguring the kernel needs to be considered. Hence, we have developed analytical performance models that cover a range of aspects such as design size, numerical precision, memory interfaces and reconfiguration overheads. These models are useful for the design-time planning and exploration of self-optimising systems, as well as the run-time execution of such optimisations. Another type of application tuning is to adjust the computation within a kernel based on external constraints. One example of this is a self-aware and self-expressive solar-powered compute unit [12]. The architecture is customised with parallel compute units that can be independently activated depending on the available power budget. During run-time, a simple geometric program is run to optimise the performance subject to the dynamically changing power constraints.

EPiCS also investigates meta-heuristics and machine learning to automate the design-space exploration of reconfigurable, self-aware applications. The previously mentioned analytical models are powerful tools for exploration and planning; however, the disadvantage is that they need to be derived manually. This can be addressed by automating the model generation with machine learning techniques. Through executing application benchmarks, we can construct a surrogate model which describes quality of the design over the parameter space. The model can also learn how design constraints are reflected through valid and invalid regions of the parameter space.

Self-verification: Finally, we want to ensure correct operation in self-aware and self-expressive systems. This is addressed through developing methods and tools for self-verification. Self-verification requires initial off-line verification testing the proposed design transformations and optimisation. This is followed by run-time verification and checking which targets dynamic optimisations and transient faults. We currently target verification of design transformations through symbolic verification and equivalence checking. Self-expression and optimisation is often facilitated by transforming a design through application tuning as described above. To evaluate whether correct functionality is maintained after tuning, we use symbolic verification. This is in contrast to numerical or logical simulation which would prove the correctness by exhaustively testing all possible input combinations. Symbolic simulation applies symbols rather than numbers or logic values to the design (e.g. a and b), and the outputs are functions of these symbolic inputs (e.g. $a + b$). In order to distinguish between symbolically different but functionally equivalent outputs ($a+b$ and $b+a$) and incorrect outputs ($a+c$), we use automatic equivalence checking. Another important aspect is verification of the reconfiguration process. This is done by modelling different configurations through virtual multiplexers: a switch between configurations is treated similar to switching with a real multiplexer. This allows us to cover reconfiguration in our symbolic simulation approach.

D. Self-Aware Networking and Middleware

To support self-awareness and self-expression in distributed computing systems, EPiCS investigates the application of these concepts to computer networks. We create an autonomous networking architecture that uses run-time reconfiguration to adapt the networking functionality to the available resources and the network traffic.

Especially for mobile devices, network characteristics change over time. They are used in buildings, in streets, in public transport systems and in the countryside. Those different environments have different characteristics with respect to throughput, reliability, required level of privacy, etc. Therefore, in contrast to the Internet architecture where the protocol stacks have to follow the strictly layered design, in the network architecture developed within EPiCS, the network functionality is split into building blocks that can be combined to provide an optimal protocol stack. Furthermore, the protocol stack can be adapted at run time and even while communicating.

In order to adapt the protocol stack, application goals, sensor input and models are required. The actual adaptation algorithm works in two rounds: first it determines the possible protocol stacks on the local node and second, it negotiates the protocol stack to be used with the destination node. Currently, the adaptation algorithm is based on simple rules, but we work on more complex algorithms that will be able to cope better with a bigger variety of situations.

As a consequence from having such flexible protocol stacks, a dynamic hardware architecture is needed, as the protocols that will be used are not known at design time. Therefore, we use the ReconOS architecture presented in section II-C as the basic architecture. We work on developing an adaptation algorithm that places network functionality optimally in either hardware or software based on the current network traffic.

The adaptation of a) the network functionality, and b) the mapping of network functionality between hardware and software provides us with a well performing communication infrastructure in terms of throughput, communication overhead and available functionality.

III. Applications and Demonstrators

The technological development in EPiCS is validated in three technology demonstrators. Throughout the project, these demonstrators are also used to refine the general concepts and technical approach.

A. Computational Finance on a Heterogeneous Compute Cluster

The first demonstrator in EPiCS explores computational finance through execution and management on a heterogeneous compute cluster. Computational finance is a field that is concerned with mathematically analysing and computationally solving problems in the financial markets. Usually, the goal is to aid, facilitate or accelerate the decision-making process in financial investment, trading or hedging. One simple example is the pricing of an American option, a financial contract that gives the owner the right but not the obligation to engage in a future transaction with a pre-arranged price up to a certain time of expiry. Determining the price of such a contract involves solving partial differential equations (PDEs). In most cases, these PDEs cannot be solved analytically and require computationally complex numerical solvers. Computational complexity arises from a requirement to solve these problems repeatedly for large portfolios with many underlying assets. It is usually desirable to run these computations as fast as possible, requiring large computational systems. Such systems deliver performance through parallelism, utilising hundreds of cores and increasingly involving heterogeneous resources such as GPUs or dedicated hardware accelerators. These systems often face many challenges relating to overall scalability, complexity and design entry. Power consumption is also an increasing concern in modern high-performance compute systems. Theses challenges in computational finance are also representative for high-performance computing in other domains such as scientific applications and engineering.

The goal of this demonstrator is to address these challenges through using reconfigurable hardware as dedicated accelerators and deploying self-adaptive implementations that can automatically adjust to changing application environments. It is well known that performance can be improved through custom-accelerators. The downside to this approach is that these accelerators often need to be hand-crafted which complicates the design process. Having to repeatedly redesign the accelerator when application parameters change exacerbates the problem. EPiCS aims to address this issue by exploring self-awareness and self-expression: this allows the application to monitor its environment and automatically adapt to changes without manual intervention. This concept is founded on the systematic exploitation of application tuning parameters, a concept that is being developed as part of the work described in section II-C.

In this demonstrator we explore application tuning through constant specialisation on reconfigurable hardware. In computational finance we can apply constant specialisation to slowly changing market parameters: compute kernels are specialised to a set of parameters that are constant for a certain amount of time depending on market conditions, and they are reconfigured when the parameters change. This is shown on the example of a financial option pricing application that uses a explicit finite difference (EFD) solver. EFD is a common method to approximate the derivative in PDEs and works by stepping through a grid of discrete values. Figure 5 illustrates a two-dimensional case for a second-order PDE. Pricing one option requires calculating the entire gird which often contains millions of values. We observe that the stencil coefficients α, β and γ are constant throughout the entire gird, and they could also be constant for longer periods depending on market conditions. Hence, we can specialise the compute kernel for these coefficients resulting in reduced area and higher performance. The kernel is automatically reconfigured whenever necessitated by changing input parameters. This self-adaptive implementation results in a performance increase of a factor of 4.7 [2].

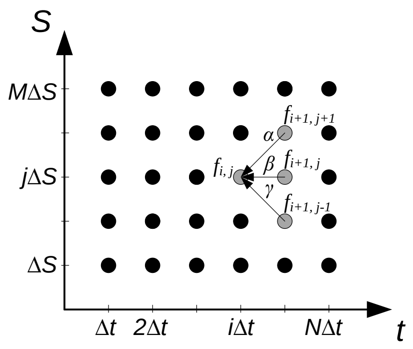

Fig. 5. Calculations of values in a finite difference grid.

Work on this demonstrator also includes developing an automated power and temperature monitoring infrastructure for compute nodes enabling power and temperature-aware techniques [14].

B. Person Detection and Tracking with Distributed Smart Cameras

The second demonstrator performs person tracking with a distributed smart camera network. The demonstrator is built mainly of available, COTS-based infrastructure. This allows us to perform very early testing of our algorithms. In the first version of this demonstrator, we use a homogeneous setup of three cameras having overlapping fields of view. We use commercially available cameras from SLR Engineering that run a Linux operating system and are equipped with a 1.6 GHz Intel Atom processor. All cameras are connected via their 100MBit Ethernet interface and have a CCD image sensor with a native resolution of 1360x1024 to capture videos. Nevertheless, we reduced the resolution to 640x480 to speed up the processing of each image. Even though our application is distributed, we use a graphical user interface to visualise the different video streams and the current status of various components of the system such as the tracker and the detector. This interface does not act as a central component for coordination and only provides visualisation.

Our initial demonstrator is able to autonomously track a single person within the three camera setup. To preserve resources within the network, only a single camera is responsible for tracking the person at any time. To achieve tracking in a distributed manner, we implemented a simple handover algorithm as described in [4]. To initiate tracking, a person or object needs to be selected by an operator, using our user interface. Based on the initial selection, a model of the person/object to be tracked is being generated. For tracking and detection we use a background appearance model to restrict the search space within the frame. In combination with the model of the object/person to be tracked we are able to reduce the amount of resources needed for tracking/detection and still get sufficient results.

During our work on this demonstrator we also developed a simulation environment to demonstrate the benefits of our handover algorithm. Furthermore we were able identify neighbourhood relationships between cameras exploiting information from the handovers between cameras. This topology information can be further exploited to improve future communication and handover between cameras. We used our simulator to show a reduction of communication of 20-40% depending on the scenario. In the next development phase we plan to increase the number of cameras in the network and extend our algorithm to learn spatial relations of cameras at run time.

C. Musical Applications on an Interactive Mobile Media System

In many musical cultures and genres there is often a large gap between those who *perform* and those who *perceive* music. In such ecosystems, the performers (musicians) *create* the music, while the perceivers (audience) *receive* the music [17]. Even though perceivers may have some control of the music creation in a concert situation, by means of cheering, shouting, etc., this only indirectly changes the musical output. The divide between performer and perceiver is even larger in the context of recorded music, which is typically mediated through some kind of playback device (CD, MP3 file, etc.). Here, the perceiver is limited to controlling only the start/stop of the playback of a pre-recorded song, and adjusting the volume of the musical sound.

The divide between those who perform and those who perceive music can be seen as a gap between an active and a passive experience of music. This clear divide is currently being challenged through various types of technologies that allow for musical experiences lying on the continuum between a traditional musician and those of a traditional listener, what we call *active music* listening. Notable examples here include karaoke machines, music keyboards with algorithmic chord/melody sequences, and computer music games like Guitar Hero and SingStar. They all allow for a certain degree of control over the musical output, while other parts are ready-made or generated algorithmically on the fly.

The development of interactive technologies and musical concepts that can work with these technologies, is what the third demonstrator relies on. This demonstrator will showcase an active music experience based on self-aware and self-expressive media device nodes, where self-awareness and self-expression form part of this feedback loop. We use the term *active music node* to describe one user, i.e. the human user and the complete system for participation in the active music environment. A device, human user, and a music engine should together be seen as assembling one active music node, as illustrated figure 6.

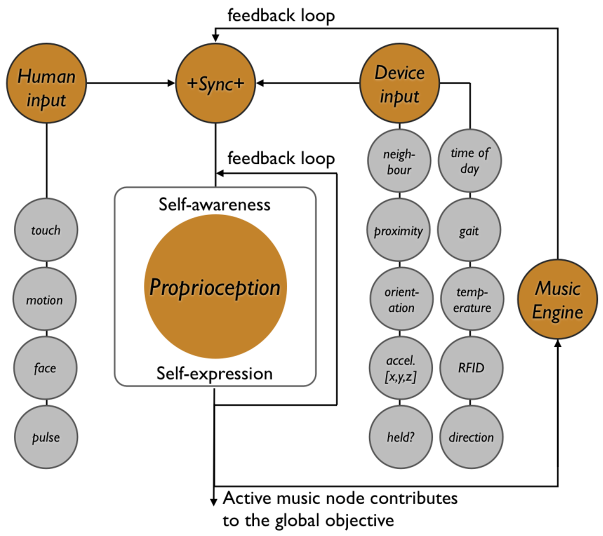

Fig. 6. Illustration of the active music demonstrator concept (one active music node).

One of the most important characteristics of our demonstrator is that humans are in the loop. A human contributes motion and embeds the device by holding and controlling it, thus becoming a part of the device itself. The device collects sensor data which is important as input to the self-aware system. Further, the music engine is the relevant component for creating self-expression in the active music node. In a setting with several active music nodes, all nodes operate in a global context and share a global objective, e.g. being part of resultant music that they might enjoy. This global objective is achieved by the contribution of all nodes in a distributed manner, and not by a master unit that controls everything like a conducted orchestra. In such a setting, the amount of information flowing between nodes also needs scaling in order to avoid hiccups or clutter that affect the music in a negative manner. It is then crucial that nodes are aware of certain characteristics about their nearest neighbours to enable possible formation of groups containing a subset of participating nodes, as necessary.

There are numerous challenges involved in creating such an active music node, and indeed a system of such nodes. At the musical level, this includes everything from low-level micro-sonic control (timbre, texture), mid-level organisation (tones, phrases, melodies) to large-scale compositional strategies (form). In addition, there are challenges related to how one or more participants can control all of these sonic/musical parameters through mappings from various forms of human input.

As part of the active music demonstrator, we have so far worked on a scenario whereby multiple active music nodes may want to have control of the same sonic/musical feature, thus resulting in a conflict as to who might be in charge of this feature at any point in time. We model this scenario as a "band" whereby active music nodes "play" together as solo artists, without there being a central controller of the music that is being generated. One node plays their solo at a time whilst holding an auction, in order to determine which node could potentially take over the playing of the solo from it as time progresses. More details on this can be found in [3].

D. Applicability Beyond EPiCS: Improved Service Provisioning in Cloud Systems

The problem of improving the quality of service offered to users of cloud systems is used to demonstrate the applicability of EPiCS' concepts to other application domains. Cloud providers in their bid to attract users promise 'elastic' service provisioning with near-infinite scalability. In reality, clouds, just like datacentres, are resource constrained and prone to failures at both node and network levels. However, in contrast to conventional datacentres, cloud providers face the problem of not being able to fully anticipate the workload patterns imposed on their infrastructure a priori. These conditions make it hard to promise high quality of services without incurring significant cost.

To alleviate this problem, novel resource allocation techniques which are more resilient to internal unexpected changes (server or network failures) and flexible to dynamics caused by external sources (e.g., spike or dwindle in workload) are needed [5]. Ongoing research in this area has investigated the use of market mechanisms engineered using the principles of self-awareness to manage interaction of computing nodes in cloud systems. Early results from simulation studies [6] indicate that the novel resource allocation method derived from this approach is more resilient to node failures in a cloud resource market. Due to the inherent decentralisation of the proposed market mechanism, it also offers capability to manage resources at the scale of cloud federations [7].

IV. Conclusion

The EPiCS project targets the challenges of developing and managing modern computing systems. Architectures are becoming increasingly parallel, heterogeneous and distributed while we also observe rising levels of dynamic behaviour in applications and environments. In EPiCS, these challenges are addressed through proprioception, self-awareness and self-expression. These concepts allow systems to reason about their state and environment, and to automatically adjust and optimise themselves. EPiCS is focused on transferring these abstract concepts from psychology to computing and communication systems. The overall approach is based on developing the theoretical concepts and foundations, and defining a requirement-driven engineering cycle for development and validation. We develop the necessary tools and infrastructure such as self-aware platforms, operating systems, networking and middleware, as well as general methods for self-optimisation and verification. This is applied to three application areas resulting in technology demonstrators that highlight how technological challenges can be practically addressed through self-awareness and self-expression. Work continues in all areas and the lessons learned from the technical approach and the demonstrators will be used to refine the theoretical basis and the overall approach throughout the project.

Acknowledgement

Research under the EPiCS project has received funding from the European Union 7th Framework Programme under grant agreement no 257906.

References

[1] A. Agne, M. Platzner, and E. Lubbers. Memory virtualization for multithreaded reconfigurable hardware. In *Int. Conf. on Field Programmable Logic and Applications (FPL)*, pages 185 –188, sept. 2011.

[2] T. Becker, Q. Jin, W. Luk, and S. Weston. Dynamic constant reconfiguration for explicit finite difference option pricing. In *Int. Conf. on ReConFigurable Computing and FPGAs (ReConFig)*, pages 176–181. IEEE Computer Society, 2011.

[3] A. Chandra, K. Nymoen, A. Voldsund, A. Jensenius, K. Glette, and J. Torresen. Enabling participants to play rhythmic solos within a group via auctions. In *International Symposium on Computer Music Modeling and Retrieval (CMMR)*, pages 674–789, 2012.

[4] L. Esterle, P. R. Lewis, M. Bogdanski, B. Rinner, and X. Yao. A Socio-Economic Approach to Online Vision Graph Generation and Handover in Distributed Smart Camera Networks. In *ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC)*, pages 1–8, 2011.

[5] F. Faniyi and R. Bahsoon. Engineering proprioception in sla management for cloud architectures. In *Software Architecture (WICSA), 2011 9th Working IEEE/IFIP Conference on*, pages 336 –340, june 2011.

[6] F. Faniyi and R. Bahsoon. Self-managing sla compliance in cloud architectures: a market-based approach. In *Proceedings of the 3rd international ACM SIGSOFT symposium on Architecting Critical Systems*, ISARCS '12, pages 61–70, New York, NY, USA, 2012. ACM.

[7] F. Faniyi, R. Bahsoon, and G. Theodoropoulos. A dynamic data-driven simulation approach for preventing service level agreement violations in cloud federation. *Procedia Computer Science*, 9(0):1167 – 1176, 2012. International Conference on Computational Science, ICCS 2012.

[8] M. Happe, A. Agne, and C. Plessl. Measuring and predicting temperature distributions on FPGAs at run-time. In *Proc. Int. Conf. on ReConFigurable Computing and FPGAs (ReConFig)*, pages 55–60. IEEE Computer Society, Dec. 2011.

[9] P. R. Lewis, A. Chandra, S. Parsons, E. Robinson, K. Glette, R. Bahsoon, J. Torresen, and X. Yao. A survey of self-awareness and its application in computing systems. In *Int. Conf. on Self-Adaptive and Self-Organizing Systems (SASO 2011)*, pages 102–107. IEEE Comp. Soc. Press, 2011.

[10] P. R. Lewis, P. Marrow, and X. Yao. A diversity dilemma in evolutionary markets. In *Int. Conf. on Electronic Commerce (ICEC 2011)*, 2011.

[11] P. R. Lewis and X. Yao. Self-awareness, self-expression and meta-self-awareness in the relevant neighbourhood selection problem. Technical Report CSR-12-02, University of Birmingham, School of Computer Science, September 2012.

[12] Q. Liu, T. Mak, J. Luo, W. Luk, and A. Yakovlev. Power adaptive computing system design in energy harvesting environment. In *Int. Conf. on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*. IEEE, July 2011.

[13] E. Luebbers and M. Platzner. Reconos: Multithreaded programming for reconfigurable computers. *ACM Trans. Embed. Comput. Syst.*, 9(1):8:1–8:33, Oct. 2009.

[14] X. Y. Niu, K. H. Tsoi, and W. Luk. Reconfiguring distributed applications in FPGA accelerated cluster with wireless networking. In *Field Program. Logic and Applications (FPL)*, pages 545–550. IEEE, 2011.

[15] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2 edition, 2003.

[16] S. Singh, J. Hogg, and D. McAuley. Expressing dynamic reconfiguration by partial evaluation. In *Field-Programmable Custom Computing Machines (FCCM)*, pages 188–194. IEEE Computer Society Press, 1996.

[17] C. Small. *Musicking: The Meanings of Performing and Listening*. Wesleyan University Press, Hanover, New Hampshire, 1998.

[18] A. van Lamsweerde. Requirements engineering in the year 00: A research perspective. In *Proc. 22nd International Conference on Software Engineering, Limerick, Ireland*, pages 5–19. ACM, 2000.