Towards a Prime Directive of SLOs

1st Victor Casamayor Pujol
Distributed Systems Group
TU Wien, Vienna, Austria
v.casamayor@dsg.tuwien.ac.at
ORCID: 0000-0003-2830-8368

2nd Schahram Dustdar
Distributed Systems Group
TU Wien, Vienna, Austria
dustdar@dsg.tuwien.ac.at
ORCID: 0000-0001-6872-8821

***Abstract*—The promises of the computing continuum paradigm motivate a paradigm change for Internet-distributed computing systems. Unfortunately, we are still far from being able to develop computing continuum systems. We try to move one step forward in the direction of the computing continuum systems by defining design phases for the interconnection of the application with its underlying infrastructure. We assume that SLOs are critical to that endeavor. Hence, we analyze its usage in the scientific literature. Based on the learnings obtained, we define 9 design phases to provide homogeneity and common behaviors in large-scale, heterogeneous, distributed, and complex systems.**

***Index Terms*—Computing continuum systems, design phases, SLO, elasticity strategies**

I. Introduction

The Computing Continuum is a novel computing paradigm [1], [2]. Its main objective is to unify all current computational tiers [3], [4], namely Cloud Computing, Fog Computing, Edge Computing, and IoT, to develop an underlying infrastructure able to accommodate applications that can benefit from the strengths of each tier while avoiding their shortcomings. For example, the Cloud provides almost unlimited computational resources, allowing us to use it for demanding tasks. Still, it is located far from its users, dealing with higher latency. On the contrary, the Edge is next to the data generation, providing lower latency to applications, but the computational resources available are constrained. Hence, the Computing Continuum paradigm foresees applications that benefit from the Edge's low latency but can leverage the Cloud in case of need.

This practical usage of the Computing Continuum raises the relevance of the application's underlying infrastructure at the same level as the application itself. The application behavior is tight to the infrastructure's characteristics, and changes in the infrastructure will severely affect the application's performance. Similarly, any change in the application will affect the topology or scale of the infrastructure in use. Simply put, these new applications benefiting from the Computing Continuum seamlessly blend with the underlying infrastructure.

This breaks with current Internet-distributed systems, where everything is defined and controlled at the application level. This bi-directional relation between the infrastructure and the application demands novel methods and techniques to model and manage distributed Computing Continuum systems.

Identify applicable funding agency here. If none, delete this.

Nevertheless, some concepts inherited from Cloud computing are expected to be helpful for the Computing Continuum. Current Cloud systems are managed through Service Level Agreements (SLAs). These are contracts between the infrastructure provider and the application developer in which the infrastructure provider guarantees service specifications. Whenever these are not satisfied, it has to pay a fine to the application developer. In practice, the application developer has to choose a Service Level Indicator (SLI) that the infrastructure provider can monitor, i.e., the availability of a specific service or the CPU load. Then, they agree on a Service Level Objective (SLO) for each SLI that constrains the acceptable values. Cloud providers will trigger (reactively or proactively) elasticity strategies to maintain the SLI within the SLO margin. Elasticity strategies are vertical, where the resources available for an instance are increased or decreased, or horizontal, where instances of the same job are created or eliminated. In distributed computing continuum systems, this can not be a dichotomy. The constraint tiers will require other strategies, such as migration and offloading, which are more challenging as they need to include new active infrastructure in the system.

We see SLOs as the mechanism to control application performance over its underlying infrastructure. SLOs provide the means to specify the application requirements and to ensure its execution according to its expected performance properly. Further, they have to be loosely coupled with candidate elasticity strategies [5]. Hence, when the expected performance is not reached, the best can be selected given the context and trigger changes in the system, which can correct the system's behavior. Simply put, we want to go beyond the business requirement for an agreement as they are now and convert SLOs into vital elements for designing complex systems that contain a dynamic, heterogeneous, multi-proprietary, and interconnected computing infrastructure.

In this article, we take a *DevOps* perspective. We aim to show the required phases to develop systems (payload/application and platform/infrastructure) that use SLOs as their key control elements. In the following, we focus on the infrastructure side. In other words, SLO-based management will appropriately link applications with their underlying infrastructure. SLOs can't be isolated constraints to a system, which are defined once the application is ready for production deployment. They need to be well aware of the system's

characteristics that are under control. Further, they have to control lower-level features, such as the energy consumption of a certain device or service, and higher-level features, such as the accuracy of a machine learning-based inference service. Hence, we need to include them in the design process without introducing extra complexity to the application developer and to develop a new set of tools and methods to enable them to properly hook the dynamic and complex underlying infrastructure to the application [6].
We elaborate on the steps needed to make SLOs first-class citizens of applications' design for the computing continuum.

In the following, Section II visits the current usages of SLOs in the scientific literature and provides background for the article. Then, Section III proposes the set of phases that we envision necessary to make SLOs first-class citizens of computing continuum systems. Finally, Section IV presents the discussion, elaborating on some loose ends, and we end with the conclusion in Section V.

II. Background

A. SLOs and applications

This section looks into the current scientific work incorporating SLOs to manage Internet-distributed applications. We aim to cover research involving the low and high-level descriptions of applications, and similarly, with the SLOs, we look at them from different abstraction levels. It is important to remark that we are missing in the literature a taxonomy combining applications and SLOs, which would ease and structure the following analysis.

Kumar et al. [7] presents a very low-level definition of applications based on the types of logical operations, i.e., a counter, a parallel counter, a matrix multiplication, and an I/O heavy application. However, the SLO described is the overall time-to-completion, a holistic measure that deviates from the low-level granularity of the applications. From our perspective, this deviation between the application and SLO definition is inconvenient. In most cases, the application will contain several of the low-level tasks explained. Hence, controlling them only with overall time-to-completion will not allow the surgical usage of elasticity strategies triggered by the SLO. One could imagine that each low-level application will benefit from a different SLO, adjusting the application performance through a specific elasticity strategy. And then, the SLO proposed would provide a holistic perspective on the application needs, which would also trigger other types of elasticity strategies.

Another angle when defining applications and SLOs in the literature is taking a workflow perspective, which is very common in ML-specific applications. Mehran et al. [8] describes two applications (road sign inspection and sentiment analysis) with all their components and relations and model them through queue theory. Then, they define ranges for overall resource requirements, allowing them to decide where to schedule workflow components. As before, there is a mismatch between the resource requirements and the workflow description, so they need to develop a complex queue theory system in order to infer where the elasticity strategy needs to take place. However, ML workflows have a fixed schema: (1) data ingestion & exploration; (2) data preparation; (3) model training; (4) model evaluation; and (5) model serving. [9] Hence, it is an opportunity to standardize SLO-based requirements for each step leveraging specific elasticity strategies instead of relying solely on an overall SLO.

Alqahtani et al. [10] developed a conceptual model to relate SLAs with IoT applications. Interestingly, they define SLOs at the infrastructure, service, and application levels. In terms of hardware (infrastructure), they conceptualize SLOs such as maximum throughput or maximum vCPU usage. At the service level, they leverage typical IoT services for an application to adapt SLOs to their needs. The services defined and the associated SLOs are, for example, a sensing service with maximum data freshness or a processing service with minimal latency. Finally, the SLO at the application level is end-to-end response time. This work shows an interesting but modest approach toward a taxonomy relating SLOs and applications by leveraging the different layers required to build Internet-distributed applications. The work does not discuss how elasticity strategies would be linked to SLOs, nor if SLOs at different levels, i.e., infrastructure and service, can be related to deal with possible inconsistencies when several SLOs are triggered.

The work from Nguyen et al. [11] is based on an augmented reality application, which is a latency-intolerant application. They developed two user-defined SLOs, maximum rejection rate, and average resource usage to control a higher level requirement, such as motion-to-display latency. Again, we find a mismatch between SLOs and the target to control. Hence they need to model the system to relate the specified SLOs with the high-level requirement. Indeed, modeling the relations between SLOs to select the elasticity strategy consistently is required. However, we imagine a more practical situation in which SLOs are defined for low- and high-level requirements.

Nigade et al. [12] work on SLO guarantee for timely Edge video analytics. They set a minimal latency to get 30 FPS as the SLO and incorporate it as the feedback of a control loop to adjust system parameters, which is their elasticity strategy. This work is interesting as they use the control loop classic structure to involve SLOs in the management process. Interestingly, they take a reactive approach to the elasticity strategies, which does not always provide the best possible results. In this regard, we have discussed in previous work (in the review process) the convenience of using a more flexible model such as the MAPE-K. This enables us to treat SLOs differently according to their specific needs. We envision components requiring fast interactions where reactivity is inevitable, while others can have more extended periods enabling proactive approaches.

Kannan et al. [13] present GrandSLAm, which leverages SLA definition per micro-service to gain detailed visibility on AI/ML application performance. They model the application architecture through a DAG and take elasticity strategy decisions with respect to the micro-service and its specific place in the DAG. Further, they have to include an end-to-end latency

SLA to enforce the expected performance, as they claim that micro-service analysis is not enough to enforce overall end-to-end latency SLA. This work is interesting as they define SLAs at different abstraction levels, keeping elasticity strategies at both levels. Unfortunately, the work is tailored to their system and the solution they developed, leaving out generalization perspectives methods.

A different perspective is taken by Furst et al. [14]. They define elastic services as those capable of elastically adjusting their characteristics to comply with their associated SLO, which is in terms of end-to-end latency. Elasticity strategies are entirely integrated into the service, obtaining adaptation capabilities such as changing the image classification model or degrading the classifier's parameters. Their approach is interesting as they describe elasticity strategies tailored to quality, which is unusual as it requires this close relation with the service. Our perspective also envisions adding this type of higher-level elasticity strategy. However, we think it is necessary to define SLOs directly addressing quality, i.e., changing the image classification model to reduce end-to-end latency also affects the quality of the service, which also needs to be considered. Further, we see the *SLO developer* as independent from the application developer, which in the work of Furst et al., both seem to be the same figure. Nevertheless, the application developer and the *SLO developer* have to cooperate, and we need bridges to join their work without the minimal overhead.

Guim et al. [15] work takes an architectural perspective for enforcing SLOs. They present a two-tier architecture where a global planner checks for the end-to-end compliance of SLOs, including overall charging and billing. And a local planner, more focused on the capabilities of the Edge platform where the service is deployed. Again, we find a different perspective on using SLOs to manage the life cycle of applications in this work. We found the usage of a 2-layer control architecture interesting, which allows them to have hierarchical relationships within the SLOs. Unfortunately, the work discusses the architecture leaving out of the scope details on the specific implementation of SLOs with the applications.

We have found the common idea of leveraging SLOs for application management purposes beyond the business-oriented perspective. However, we have also found that each research presents a different approach to using SLOs. Similarly, their granularity and the relation between SLOs and applications or services vary from each work. Interestingly, the variety we found hints towards the need to define SLOs at different abstractions, which must be aligned with the service or application granularity they manage. Also, we found that defining relations between SLOs is needed to better hook applications with underlying infrastructures. We think working on processes and definitions at the design level is necessary to enable the future computing continuum and its manifold of applications.

B. Background concepts

1) Markov Blanket: The Markov Blanket of a target random variable is all those variables that provide the target conditional independence from any other random variable [16]. This implies that the information obtained from the Markov Blanket variables is enough to infer the target random variable state perfectly. Hence, they provide filtering to the target variable, so there is no need to evaluate variables that are not part of its Markov Blanket. Consider that when the set of variables is minimal, it is called Markov Boundary. For simplicity, we will call it Markov Blanket as we prefer working with the minimal set, but it is not a strong requirement.
Interestingly, Markov Blankets are also used in neuroscience as a mathematical artifact to separate the brain from the external environment [17]–[19]. This research line also has an ontological perspective and classifies Markov Blanket nodes as sensing, which are influenced by the environment and influence the target variable, and as acting, which is influenced by the target variable and influences the external environment. In our research, we take from both sides, aiming to apply the benefits of this mathematical concept to the distributed computing continuum systems [20], [21].

2) Causal inference: Causal inference is a technique to infer causal relations and their consequences between variables. It is a technique developed by Judea Pearl [22]. Is it a data-based technique, it uses data to extract the relations between variables. Interestingly, three causal rungs are defined depending on the type of query that you aim to solve with the data. The first one is observational, which is the information that you can obtain from data without influencing it. The second one is interventional, so you need to act on the data and learn the consequences of the performed action. The third rung is the counterfactual, which is able to make hypotheses about things that did not happen. Obtaining data to solve this type of query is not obvious and might not be possible. The explainability capacity of the method, as well as its novel mathematical framework, make this technique very convenient for distributed computing continuum systems.

3) DeepSLO: DeepSLOs are a construct relating to Service Level Objectives (SLOs). In large, distributed, and interconnected applications, SLOs can specify requirements for a single element, and they can't be related or influenced by any other system component. From our perspective, this is a shortcoming for SLOs. Hence, we define DeepSLOs as the construct able to relate SLOs regardless of their abstraction level. Indeed, relating different abstraction levels might require intermediate steps. For example, connecting GPU usage with inference quality requires at least an intermediate connection which can be the inference model used. For more details, check our previous work [2], [21].

III. Design phases

This section delineates the phases we foresee to link the application with the underlying infrastructure through SLOs. There are two considerations to be clarified before diving into the design phases. First, the amount of work on these phases

is significant. Hence we think a new figure, i.e., *the SLO developer*, is needed. This work must be done in parallel with the application development, and we assume collaboration and a fluid exchange of information. This brings us close to the second consideration, which is in terms of the initial requirements for the process. The logical system architecture of the applications and the overall system requirements are needed as inputs. These are the minimum requirements to bootstrap the design phases.
At design time, the process consists of 9 phases, as seen in Figure 1. They are briefly introduced next:

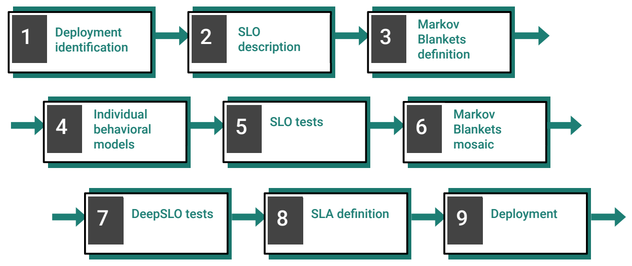

Fig. 1. Design phases for SLO management of computing continuum applications

1) **Deployment identification** defines the *functional infrastructure deployment* for the application.
2) **SLO description** describes SLOs at infrastructure, service, and application levels.
3) **Markov Blankets definition** structure system variables to get complete definition and control for each SLO.
4) **Individual behavioral models** enrich each SLO with causality-based behavioral models linking SLO states with elasticity strategies.
5) **SLO tests** certify the Markov Blanket definitions together with the behavioral models.
6) **Markov Blankets mosaic** develops relations between Markov Blankets.
7) **DeepSLO tests** assess deep (hierarchical) relations within the Markov Blankets mosaic.
8) **SLA definition** validates and agrees with infrastructure providers on the system's control implementation.
9) **Deployment** conducts the actual system development by finally integrating the application with its underlying infrastructure.

See Figure 2 for a summary of phases 1 and 2; Figure 3 for phases 3 to 5; Figure 4 for phases 6 and 7; and finally, Figure 5 for phases 8 and 9.

Case study - Aid system for urban mobility

A case study will be used during this section to exemplify and help understand each of the phase descriptions. The application example is similar to the one presented in [2]. However, instead of checking if drivers are using a phone, we want to depict a more distributed application through an entire city, which understands and predicts the city mobility behavior to provide near-real-time guiding aid to drivers and pedestrians, and, eventually, to facilitate autonomous vehicles. We have changed the focus of the case study to emphasize geographical distribution, low-latency requirements, and heterogeneity of devices.
The simplest example of this application would be a citizen planning a trip from point A to B of the city. Then, the system recommends alternative routes and travel modes depending on the city's predicted mobility situation for a planned time. However, a more demanding example of this application would be a near real-time assessment of the city roads' status to optimize autonomous mobility by providing tailored routes for each autonomous vehicle.

A. Deployment identification

This initial phase defines the infrastructure components that can be required for the system. We see three levels of infrastructure definition: (1) the *minimal infrastructure deployment*, (2) the *functional infrastructure deployment*, and (3) the *complete infrastructure deployment*. We expect this phase to provide the description of the *functional infrastructure deployment*. This consists of identifying where the logical components of the system could be deployed, and functional expresses that the only constraint taken is that the system has to perform accordingly to the overall requirements, but there is no other optimization required. Conversely, the *minimal infrastructure deployment* is an optimization iteration over the functional to limit the infrastructure components, which can be interesting to reduce costs or energetic footprint. And the *complete infrastructure deployment* can be achieved once all design phases have been covered, identifying all infrastructure components that eventually can be part of the system.
In order to successfully define the *functional infrastructure deployment* the logical system's architecture needs to be mapped into actual infrastructure components. For this process, we foresee collaboration from the *application developer* on assessing hardware components for each application service. It is important to remark that no optimization is expected at this early stage. Further, several hardware options can be considered per service, which can be discarded later or kept as backup configurations within the *complete infrastructure deployment*. In any case, the *SLO developer* will specify the hardware components required. To that end, a taxonomy of the available hardware components per provider is needed to finalize this phase. Given the characteristics of the computing continuum infrastructure, we envision a scattered landscape of infrastructure providers: Cloud, Fog, Edge, IoT, network, etc. Additionally, one can imagine 3rd party service providers including their hardware within the package. Simply put, if you need an ML model for your application, it can be more convenient to use existing and trained models that a provider can offer with their own infrastructure. Nevertheless, we want to emphasize the need for a tight relationship with infrastructure providers to obtain solid progress in the design phases.

Case study: For the application depicted the *functional infrastructure deployment* should include IoT sensors such as

cameras, radars, and traffic lights. Each cluster of sensors would require processing units (Edge nodes), to reduce communication latency and to pre-process the data, performing initial inferences. Then, the information could be aggregated in larger computing units (Fog nodes) to perform heavier computations and to redirect data to the application control center and to the Cloud for storage, analytics, and long-running ML jobs. Further, the geo-location of the autonomous vehicles can require a special-trusted connection with them, and further, the broadcasting of routes to these vehicles also requires special network capabilities to ensure near real-time connectivity for the entire city.
Considering that the infrastructure will not be built for this specific application, we are set in a multi-tenant environment. Further, in some situations, the usability of the device can be complete, i.e., dedicated Edge nodes for data pre-processing, but in other cases, a limited view can only be provided, i.e., the video feed from street cameras can be used only if processed in place with strict privacy policies. In brief, there's a manifold of different possibilities and these need to be explored at the start of the system development.

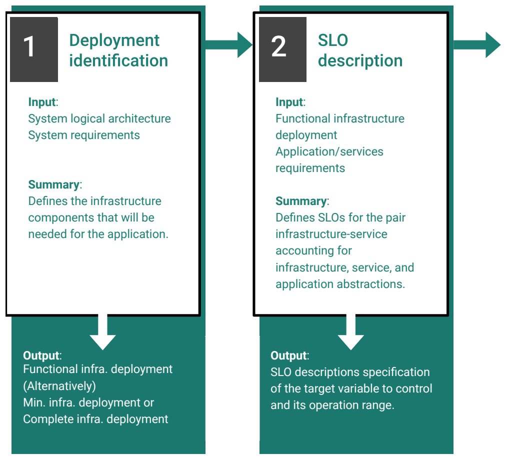
1
Deployment identification
Input:
System logical architecture
System requirements
Summary:
Defines the infrastructure components that will be needed for the application.
Output:
Functional infra. deployment
(Alternatively)
Min. infra. deployment or
Complete infra. deployment
2
SLO description
Input:
Functional infrastructure deployment
Application/services requirements
Summary:
Defines SLOs for the pair infrastructure-service accounting for infrastructure, service, and application abstractions.
Output:
SLO descriptions specification of the target variable to control and its operation range.

Fig. 2. Phase 1: Deployment identification & Phase 2: SLO description

B. SLO description

This is a key phase for the design process as it determines the SLOs as the hook between the application and the infrastructure. Hence, it takes the *functional infrastructure deployment*, and for each pair, service hardware defines at least 1 SLO to control the hardware and 1 to control the service. Further, SLO at a coarser granularity has to be defined, for instance, at the workflow or application level, depending on the most convenient approach for the system. Defining an SLO involves selecting a key measurable metric for the component performance and providing a range within its values to ensure that performance.
At this phase, the *SLO developer* has to interact with the *application developer* and the infrastructure providers. Basically, finding this key metric for the component performance can be challenging, hence the people developing the service can shed some light on what they would need, and similarly, the infrastructure provider can offer some SLO descriptions that are convenient for their hardware. In the end, the *SLO developer* has to choose the best SLO for each piece, and having the overall picture of it can ease the process by eliminating redundancy or using complementary metrics. Further, the collaboration with the *application developer* becomes more interesting as a key measurable metric for a service can require intentionally offering that metric for the application management, which is something that in current developments does not exist. Simply put, a key metric for the ML model can be the inference latency, but it can also be the prediction accuracy, but the second's observability needs to be specifically implemented on the service.

Case study: We can imagine SLOs governing IoT devices or services, such as continuous *data availability* from the cameras. Edge nodes having a maximum *processing time*. We can also increase the abstraction level of SLOs to control Cloud *storage costs*, or to provide *privacy guarantees* through an SLO when processing vehicles' geo-location at fog nodes. Privacy guarantees are tricky to address as an SLO; simply put, how can they be measured? An option would be as follows, privacy guarantees can be translated to privacy policy enforcement, and hence, the SLO could be the complete enforcement of all privacy policies. These are only 4 examples at different abstraction levels, from processing time or service availability to cost control or privacy protection. Key to this phase is the capacity to conceptually enlarge the possibilities of SLOs as control metrics for the system.

C. Markov Blankets definition

Once the key metric to control the component has been identified, this third phase benefits from a mathematical construct, the Markov Blanket, to identify all other metrics and configurations that affect the selected key metric. Given the characteristics of computing continuum systems, explaining the variation of the key metric can be challenging. Hence, defining its Markov Blanket identifies those system variables (sensory variables) that influence it, as well as those configurations (acting variables) that can affect its behavior, e.g., configuring the usage of an old ML model can accelerate the accuracy degradation process. The Markov Blanket ensures that our key metric is independent of all other system variables and configurations except the ones included. This simplifies any analysis, easing the process of understanding the key metric behavior.
Obtaining the Markov Blanket for each SLO previously defined is a complex task. Indeed, previous knowledge of the task can ease the process. However, this phase will most likely become an iterative process together with the next 2 phases. The *SLO developer* has to initially provide an educated guess of the Markov Blanket variables, which in case of possessing the system's performance data guiding the selection. Several algorithms can analyze data to determine the Markov Blanket of a target variable [23], [24]. However, data might not be

available at the early design stages.
There are two relevant considerations to describe in this phase. The first concerns the variables of the Markov Blanket. Acting variables can influence the behavior of the key variable (SLO), however, they are completely dependent on the system's design. In other words, if the design does not include any configuration or action capacity over a key metric, this will not appear on the Markov Blanket, and the system will work as expected without it. This is the opposite behavior with respect to the sensory variables, which their omission will not free the system behavior of their influence, leading to a complex situation where the performance's explainability can be severely hampered. This behavior enforces the need of iterating over this phase to ensure the consideration of all sensory variables.
The second is about the idea of dynamic Markov Blankets, these would be able to automatically change when a configuration on the system or an external event changes its behavior. Currently, there are algorithms that based on data can define the Markov Blanket, hence, it is not difficult to imagine an algorithm that based on new data takes a previous Markov Blanket configuration and updates it. This is an interesting perspective, however, the interconnectedness of these systems can propagate changes requiring large and deep reconfiguration with unpredictable results.

Case study: Once an SLO has been selected, e.g., Edge nodes processing time or privacy guarantees for autonomous vehicles, this phase has to determine the specific system variables that directly affect (sensing variables) the SLO, similarly, it has to identify which are the configuration parameters that can change the SLO behavior (acting variables). In the case of having system data from prototypes, datasets, or previous experiences structure learning algorithms can be used to automatically detect which are the components of the Markov Blanket. If not, the iterative process described in this article needs to be pursued. If we take the Edge nodes processing time in charge of pre-processing cameras data, we can select several sensing variables such as stream FPS, image resolution or number of incoming streams. Concerning the acting variables one can think about limiting frame rate or resolution, spinning a new container on the same facilities, or splitting the processing pipeline to finish it at the fog nodes.
Privacy guarantees for autonomous vehicles, or the complete enforcement of privacy policies, can have as sensing variables the number of vehicles considered, the location(s) where policies are enforced, or the overall time required for their guarantee. Acting variables could consider the specific algorithms used, their execution environment, or the total time that the sensitive data is stored.

D. Individual behavioral models

In the previous phase, each Markov Blanket included acting variables, described as configuration options to affect the SLO behavior. In Cloud computing, these would be the elasticity strategies. However, elasticity strategies are currently unconsciously linked to the scaling of computing resources, but

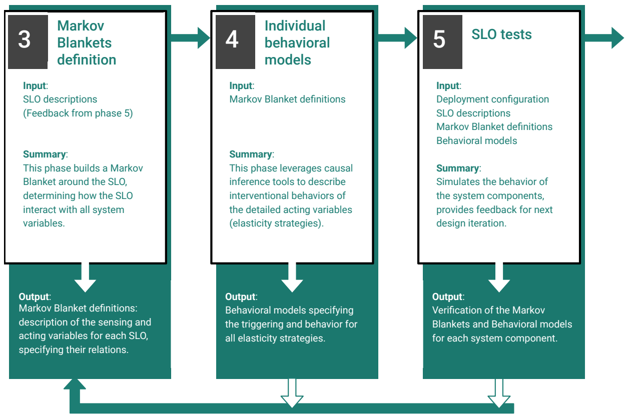
3
Markov Blankets definition
Input:
SLO descriptions
(Feedback from phase 5)
Summary:
This phase builds a Markov Blanket around the SLO, determining how the SLO interact with all system variables.
Output:
Markov Blanket definitions: description of the sensing and acting variables for each SLO, specifying their relations.
4
Individual behavioral models
Input:
Markov Blanket definitions
Summary:
This phase leverages causal inference tools to describe interventional behaviors of the detailed acting variables (elasticity strategies).
Output:
Behavioral models specifying the triggering and behavior for all elasticity strategies.
5
SLO tests
Input:
Deployment configuration
SLO descriptions
Markov Blanket definitions
Behavioral models
Summary:
Simulates the behavior of the system components, provides feedback for next design iteration.
Output:
Verification of the Markov Blankets and Behavioral models for each system component.

Fig. 3. Phase 3: Markov Blankets definition. Phase 4: Individual behavioral models & Phase 5: SLO tests

in the current scenario, we aim to expand their possibilities. Besides the elasticity capacity, acting variables can modify the quality of the service by changing data granularity or the machine learning model, but they can also change the network communication protocol or migrate the service to another infrastructure component. This phase must define, for each Markov Blanket, a behavioral model. This can be simplified as the set of rules that given a specific configuration of the Markov Blanket variables and the SLO, an acting variable is activated. This leads to a reconfiguration of the SLO, which will set it back to its expected range. The behavioral model has a double objective, first, selecting the acting variable that will adjust the SLO values, and second, predicting which will be the state of the Markov Blanket variables after the configuration change has been conducted. Remarkably, the second objective enables optimization capabilities to the Markov Blanket behavior, given that several acting variables can set the SLO back to its expected range but each of them leaves the state of the Markov Blanket differently.
Behavioral models can be defined using causal inference methods. This technology has two main benefits; first, it can produce explainable results, which would ensure that the behavioral models used are accountable. And second, the model can describe interventional causal relations, not only observational causal relations between variables, i.e., the model's behavior if it is not externally affected, which usually is the scope of the available data. This means it can explain how variables will behave after the activation of an acting variable. Developing these models require data, further, interventional queries on data require that these are also part of the analyzed data. Hence, simulation can be the necessary tool for gathering the data to understand SLO behavior. Regardless of this being our preferred method, one can easily see techniques such as Reinforcement learning [25] or Active inference [26], [27] work in a simulation environment to extract similar knowledge with causal inference. Interestingly, some research tries to combine reinforcement learning with causal inference [28].

Case study: We can take now one of the previously defined Markov Blankets, e.g., Edge nodes processing time, we can consider that if the time goes above a certain threshold and less than 3 containers are working on it, then the best is to add up to 3 instances. However, if there are 3 containers already on the task, the best way to reduce the processing time is by reducing the FPS to half. This behavior could have been obtained from experience. If no experience is available, then data from the system is required to take a causal inference approach or a simulation environment to benefit from reinforcement learning-like techniques.

E. SLO tests

At this point, we have defined for each combination of service and infrastructure an SLO with its Markov Blanket and its behavioral model able to control that system (service and infrastructure) component. However, knowledge of the system can be limited, hence this phase aims at testing the reliability of the design decisions. Further, the data obtained from testing can be considered to improve the two previous phases (*Markov Blankets definition & Individual behavioral models*). Both phases require data to ensure their validity and the data from the tests can be re-injected in the process to refine the decisions. Last, an ultimate objective of this phase is to certify to the infrastructure provider the behavior of the SLOs defined easing the future contractual agreement. We envision a large and scattered landscape of infrastructure providers for computing continuum systems. This will change the current Cloud paradigm and we expect that cost reduction will not be the only motivating premise. In that regard, we can foresee infrastructure providers developing simulation tools to enable users to test and verify their applications. This might currently seem unrealistic, but, large system engineering projects require many providers, and their final selection is based on a weighted score concerning several categories. When dealing with materials, you get samples of them; when dealing with software, you can get a demo of it, etc. Hence, you might get simulation tools to verify your design decisions in an appropriate environment when dealing with infrastructure providers. In any case, we assume that a simulation tool able to test the design exists, and hence, each Markov Blanket is tested until all behaviors can be properly explained leveraging these 3 stages (Markov Blanket $\rightarrow$ Behavioral model $\rightarrow$ Simulation) iterative process.

Case study: At this stage, we would have all Markov Blankets with their specified behaviors, and it would be the moment of properly testing them. This process should be done in a simulation environment due to scale and costs reasons. In the depicted case study, where there is no prior knowledge of the system's behavior, the initial configuration and the described behaviors are inaccurate. Hence, we would first use the simulation environment to properly define the Markov Blankets, ideally becoming a Markov Boundary (the minimal set of variables that affect the target one). Once the Markov Blankets (we keep this naming for simplicity) are correctly specified, then the behavioral models can be refined, newly discovered, or completely changed. In any case, this process can require more than one iteration over the simulation, in the end, it might depend on the initial discrepancy between the Markov Blankets and the simulated system behavior.

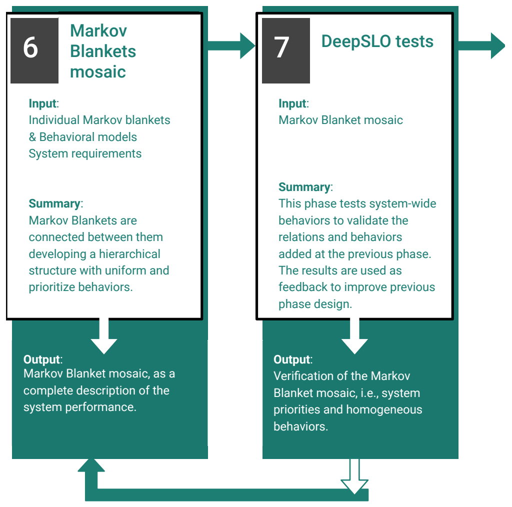
6
Markov Blankets mosaic
Input:
Individual Markov blankets & Behavioral models
System requirements
Summary:
Markov Blankets are connected between them developing a hierarchical structure with uniform and prioritize behaviors.
Output:
Markov Blanket mosaic, as a complete description of the system performance.
7
DeepSLO tests
Input:
Markov Blanket mosaic
Summary:
This phase tests system-wide behaviors to validate the relations and behaviors added at the previous phase. The results are used as feedback to improve previous phase design.
Output:
Verification of the Markov Blanket mosaic, i.e., system priorities and homogeneous behaviors.

Fig. 4. Phase 6: Markov Blankets mosaic & Phase 7: DeepSLO tests

F. Markov Blankets mosaic

Before starting this phase, each individual component of the system governed by an SLO is working as expected. However, the system's behavior goes beyond the aggregation of its parts. This phase relates all specified Markov Blankets, from the lower-level infrastructure to the higher-level business SLOs, developing the system's DeepSLOs. It is important to clarify that these relations are not about connecting all Markov Blankets with all Markov Blankets, but to relating (and uniforming) system behaviors when dealing with the same abstraction level, and incorporating directives from application and business purposes to different abstraction levels. Further, such relations aim to provide priorities to the activation of acting states when SLOs deviate from their expected range. Simply put, if an SLO deviates from range, now that Markov Blankets are connected, the acting states of others can also influence the deviated SLO. Hence, we need to know which one to activate. Similarly, higher-level SLOs can be better suited to formalize activation priorities. There are two considerations worth discussing when tying together Markov Blankets of different abstraction levels (the following is less prominent at the same abstraction level). The first is about the semantics, each abstraction can use its own semantics conveniently for its purpose, however, when relating them it has to be ensured that both Markov Blankets properly understand the information passed. The second is about the temporal scales of the Markov Blankets, intuitively higher-level abstraction Markov Blankets will have longer control periods than lower-level, i.e., the infrastructure-related metrics will have shorter frequencies than the metrics related to services, such as performing an ML inference. This will need to be considered when relating the Markov Blankets, and

it will, most probably, enforce a top-bottom hierarchy. All in all, in this phase, the system becomes a mosaic of Markov Blankets with a hierarchical structure able to make decisions that all components take similarly obtaining a smooth and unified system behavior.
Successfully developing the Markov Blanket mosaic is a complex task. Hence, we also envision an iterative process with the following phase, *DeepSLO tests*. In the case of relating Markov Blankets of the same abstraction level, we expect to find sensory and acting states that are the same, or at least, very related. Hence, we will use them to relate the behavioral models of each Markov Blanket. A different case is when relating Markov Blankets of different abstraction levels. In that situation, the most common connections will be through the acting nodes. These can develop configurations that relate Markov Blankets from different abstraction levels and intentionally affect the system considering the relation created.

Case study: In this phase, we are taking all Markov Blankets and their behaviors, which have been properly validated, constructing system-wide homogeneous behaviors, and prioritizing high-level strategies over lower levels by means of hierarchical relations.
Continuing with the two Markov Blankets defined (Edge nodes processing time and privacy guarantees), we have to prioritize privacy guarantees over the Edge nodes' processing times. We can envision both services using Edge nodes and we can also relate the processing time with the policy enforcement system capacity. Hence, it would be convenient to prioritize the reduction of FPS and/or the image quality before moving the privacy guarantees process away from the Edge where leakages can be more dangerous. Similarly, policies that incur into lower costs can be preferred through the Markov Blanket mosaic instead of quality-preserving directives, e.g., reducing data granularity.

G. DeepSLO tests

Now that the system behavior is specified by constructing the Markov Blanket mosaic, it is required to verify its correct performance. Hence, these tests, most likely in a simulation environment, need to show that the system has unified and clear directives showing coordinated behavior. As suggested before, achieving such behavior is assumed to be complex due to the scale and interconnectedness of these systems. Hence, we also see this phase as the feedback loop to correctly generate the Markov Blanket mosaic through an iterative process with the previous phase. We envision that the simulation environment used in this step could be a joint effort of the infrastructure providers' community. Similarly, as we have argued before, the large and scattered landscape of providers will need to break some of the current barriers, e.g., interoperability issues, in order to enable computing continuum systems. Hence, building tools that are able to simulate together providers from different tiers can homogenize interoperability and semantics and provide the application developers with the right environment to have a realistic end-to-end simulation of their systems. Simply put, the relations created will need to be triggered by the simulation environment, showing, eventually, the expected system behavior. In any case, the process has to generate an input for the feedback loop.

Case study: Assuming no prior knowledge of the system's behavior, this phase and the previous one will require iteration to properly construct the Markov Blanket mosaic and its internal DeepSLO. Following the case study, simulating the system we could observe that prioritizing lower cost policies through the Markov Blanket mosaic leads to undesired effects such as difficulties in timely enforcing privacy policies, hence, risking privacy guarantees. From that learning, the lower cost policies can be bounded to specific Markov Blankets of the mosaic, and let the other prioritizing not a minimal cost but a range.
Similarly, as in phase 5 (SLO tests), the extracted knowledge in simulation is used to refine policies and develop meaningful links between Markov Blankets. Once the behaviors are obtained in the simulation environment, the Markov Blanket mosaic can be finalized and validated.

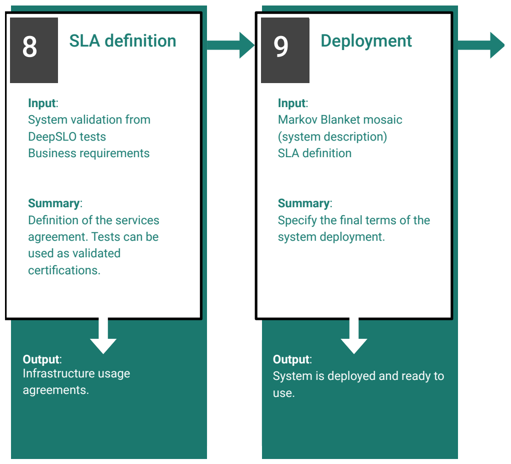
8
SLA definition
Input:
System validation from DeepSLO tests
Business requirements
Summary:
Definition of the services agreement. Tests can be used as validated certifications.
Output:
Infrastructure usage agreements.
9
Deployment
Input:
Markov Blanket mosaic (system description)
SLA definition
Summary:
Specify the final terms of the system deployment.
Output:
System is deployed and ready to use.

Fig. 5. Phase 8: SLA definition & Phase 9: Deployment

H. SLA definition

This phase aims at agreeing with all infrastructure providers on the system's expected behavior and the economic consequences if the infrastructure does not perform as expected. Interestingly, in such a complex environment, many contractual clauses that we can't still conceive can be expected depending on the type of infrastructure, e.g., limited infrastructure tenancy, maintenance scheduling for automated vehicles, etc. Nevertheless, it is a phase mostly driven by business issues, however, conflict with the design decisions can arise, and hence, it is convenient a proper interaction between the *SLO developer* and the infrastructure providers from the beginning to avoid time delays and extra costs. Related to the previous test phases, we envision that the quality of the simulation tests performed at the Markov Blanket and system level is enough to ease the contractual agreement using the obtained results as the basis.

a) ***Case study:*** At this phase, the aid system for urban mobility application design is ready. Hence, agreements with all infrastructure and service providers need to be signed. We are assuming that the results from the simulation environments are proof of the system behavior that all parties accept. Otherwise, the process would have required specific steps for each of the providers. Anyhow, we can imagine interactions with all of them: the street cameras, the autonomous vehicles, Edge nodes, Fog nodes, the network, etc. Hence, their proper inclusion during the design process is needed.

I. Deployment

We add deployment as a design phase because there are still decisions to be made that can affect the system's behavior. Up to now, we have assumed that the design has been done for the *functional system deployment*. However, depending on the application's reach or requirements, one might want to deploy the *complete system deployment* or several instances of the *minimal system deployment*. Further, it is also conceivable to deploy some non-critical components first to ensure that the system basics are properly *up & running* before deploying the rest of the system. In any case, these decisions are concurrent with the SLA definition, as the agreements can be dependent on it.

Case study: At this stage, the Markov Blankets governing the application SLOs as well as the associated services would be deployed in the computing continuum infrastructure, and the application would be available for users.

IV. Discussion

This paper provides prime directives for a new methodology to design distributed computing continuum systems. It is essential to clarify that community agreement, new methods, and tools are required to develop this vision further. Regardless, we believe it is important to start pushing together in one direction to make future computing continuum systems feasible. Having said that, we want to shed some light on a few topics that have been slightly covered in the previous section.

In current development approaches, the software development is previous and independent of the final computing infrastructure in which it will be running. The new computing continuum paradigm requires a tight relation between the software and its underlying infrastructure. Hence, we propose that both tasks, software development, and infrastructure management, work together. This way better synergies and performances can be obtained.

The *SLO developer* is envisioned as a new member of an application design team. The complexity and heterogeneity of the underlying infrastructure of distributed computing continuum systems preclude adding their management responsibility to software developers. We are not saying that software developers can not be *SLO developers*, just that these roles need to be differentiated. Further, we assume that the scattered and large-scale landscape of providers will further force the need for the *SLO developer* role and new techniques to orchestrate infrastructure from several providers.

In the provided directives, we are assuming that all SLOs involved are determined by the application developer who will be in charge of its performance. However, we do not discard that infrastructure providers can also offer a set of SLOs that the application developers can leverage. An interesting derivative is on the elasticity strategies, which are currently linked to the SLO, but we claim that they need to be completely decoupled [5], as there is going to be a manifold of options to align SLOs again. Hence we expect infrastructure providers not only to offer SLOs but also to offer elasticity strategies independently. This way, the application developer can build the system accordingly to their principles.

We have only briefly approached this topic, but we believe that, with the appropriate environment, infrastructure providers can also offer services with their infrastructure and SLOs. We can foresee many systems making similar decisions when thinking about the Edge and IoT. For example, street cameras could be used for traffic monitoring, AR/VR applications, crowd control, and many others. But it is not sustainable to have specific hardware for each application. We are in a multi-tenant setting. Hence, the camera owner, instead of offering access to the camera, could offer the video stream in different formats and with associated SLOs to satisfy any of the applications. Similar reasoning can be given when performing inference at the Edge so that the inference results can be given as a service, alleviating the application of training, validating, testing, and updating models. Indeed not all applications will leave the inference engine to others, but depending on the application's core business, that can benefit them.

Last, we assume that the following life cycle phases of these systems, e.g., run-time or end-of-life, will also require specific processes and technologies to deal with the dynamism and complexity of computing continuum systems. However, these are out of the scope of this article.

V. Conclusion

This article has shown and discussed the prime directives for SLOs to convert them into key elements for designing computing continuum systems. We have reviewed the usage in the scientific community of the SLOs and their relations with the application. Based on this analysis, we have provided 9 design phases to leverage SLOs as the hook between the application and the underlying infrastructure. We assume that Markov Blankets and Causal inference are enablers for these types of systems, and we envision methods to include them in the design process. Further, as a case study, we have provided an example where we have tried to shed some light on the meaning of the described phases. Overall, we are convinced that this is a required step to make computing continuum systems a reality soon.

References

[1] F. Firouzi, B. Farahani, and A. Marinšek, "The convergence and interplay of edge, fog, and cloud in the AI-driven Internet of Things (IoT)," *Information Systems*, p. 101840, July 2021. Publisher: Pergamon.

[2] S. Dustdar, V. Casamajor Pujol, and P. K. Donta, “On distributed computing continuum systems,” *IEEE Transactions on Knowledge and Data Engineering*, 2022. Publisher: IEEE Computer Society.

[3] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, and J. P. Jue, “All one needs to know about fog computing and related edge computing paradigms: A complete survey,” *Journal of Systems Architecture*, vol. 98, pp. 289–330, Sept. 2019. Publisher: North-Holland.

[4] Costa Breno, B. J. Jr., d. C. L. Rebouças, and A. A. P. F., “Orchestration in Fog Computing: A Comprehensive Survey,” *ACM Computing Surveys (CSUR)*, vol. 55, pp. 1–34, Jan. 2022. Publisher: ACM PUB27 New York, NY.

[5] S. Nastic, A. Morichetta, T. Pusztai, S. Dustdar, X. Ding, D. Vij, and Y. Xiong, “SLOC: Service level objectives for next generation cloud computing,” *IEEE Internet Computing*, vol. 24, pp. 39–50, May 2020. Publisher: Institute of Electrical and Electronics Engineers Inc.

[6] W. Tärneberg, E. Fitzgerald, M. Bhuyan, P. Townend, K.-E. Årzén, P.-O. Östberg, E. Elmroth, J. Eker, F. Tufvesson, and M. Kihl, “The 6G Computing Continuum (6GCC): Meeting the 6G computing challenges,” in *2022 1st International Conference on 6G Networking (6GNet)*, pp. 1–5, July 2022.

[7] R. Kumar, M. Baughman, R. Chard, Z. Li, Y. Babuji, I. Foster, and K. Chard, “Coding the Computing Continuum: Fluid Function Execution in Heterogeneous Computing Environments,” in *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 66–75, June 2021.

[8] N. Mehran, Z. N. Samani, D. Kimovski, and R. Prodan, “Matching-based Scheduling of Asynchronous Data Processing Workflows on the Computing Continuum,” in *2022 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 58–70, Sept. 2022. ISSN: 2168-9253.

[9] I. Syrigos, N. Angelopoulos, and T. Korakis, “Optimization of Execution for Machine Learning Applications in the Computing Continuum,” in *2022 IEEE Conference on Standards for Communications and Networking (CSCN)*, pp. 118–123, Nov. 2022.

[10] A. Alqahtani, Y. Li, P. Patel, E. Solaiman, and R. Ranjan, “End-to-End Service Level Agreement Specification for IoT Applications,” in *2018 International Conference on High Performance Computing & Simulation (HPCS)*, pp. 926–935, July 2018.

[11] C. Nguyen, C. Klein, and E. Elmroth, “Elasticity Control for Latency-Intolerant Mobile Edge Applications,” in *2020 IEEE/ACM Symposium on Edge Computing (SEC)*, pp. 70–83, Nov. 2020.

[12] V. Nigade, R. Winder, H. Bal, and L. Wang, “Better Never Than Late: Timely Edge Video Analytics Over the Air,” *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, pp. 426–432, Nov. 2021. Conference Name: SenSys ’21: The 19th ACM Conference on Embedded Networked Sensor Systems ISBN: 9781450390972 Place: Coimbra Portugal Publisher: ACM.

[13] R. S. Kannan, L. Subramanian, A. Raju, J. Ahn, J. Mars, and L. Tang, “GrandSLAm: Guaranteeing SLAs for Jobs in Microservices Execution Frameworks,” in *Proceedings of the Fourteenth EuroSys Conference 2019*, EuroSys ’19, (New York, NY, USA), pp. 1–16, Association for Computing Machinery, Mar. 2019.

[14] J. Fürst, M. Fadel Argerich, B. Cheng, and A. Papageorgiou, “Elastic Services for Edge Computing,” in *2018 14th International Conference on Network and Service Management (CNSM)*, pp. 358–362, Nov. 2018. ISSN: 2165-963X.

[15] F. Guim, T. Metsch, H. Moustafa, T. Verrall, D. Carrera, N. Cadenelli, J. Chen, D. Doria, C. Ghadie, and R. G. Prats, “Autonomous Lifecycle Management for Resource-Efficient Workload Orchestration for Green Edge Computing,” *IEEE Transactions on Green Communications and Networking*, vol. 6, pp. 571–582, Mar. 2022. Conference Name: IEEE Transactions on Green Communications and Networking.

[16] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1988.

[17] M. Kirchhoff, T. Parr, E. Palacios, K. Friston, and J. Kiverstein, “The Markov blankets of life: autonomy, active inference and the free energy principle,” *Journal of The Royal Society Interface*, vol. 15, Jan. 2018. Publisher: The Royal Society.

[18] I. Hipolito, M. Ramstead, L. Convertino, A. Bhat, K. Friston, and T. Parr, “Markov Blankets in the Brain,” *Neuroscience and Biobehavioral Reviews*, vol. 125, pp. 88–97, June 2020. arXiv: 2006.02741 Publisher: Elsevier Ltd.

[19] E. R. Palacios, A. Razi, T. Parr, M. Kirchhoff, and K. Friston, “On Markov blankets and hierarchical self-organisation,” *Journal of Theoretical Biology*, vol. 486, p. 110089, Feb. 2020. Publisher: Academic Press.

[20] V. Casamayor Pujol, P. Raith, and S. Dustdar, “Towards a new paradigm for managing computing continuum applications,” in *IEEE 3rd International Conference on Cognitive Machine Intelligence, CogMI 2021*, pp. 180–188, Institute of Electrical and Electronics Engineers Inc., 2021.

[21] V. Casamayor Pujol, A. Morichetta, I. Murturi, P. Kumar Donta, and S. Dustdar, “Fundamental Research Challenges for Distributed Computing Continuum Systems,” *Information*, vol. 14, p. 198, Mar. 2023. Number: 3 Publisher: Multidisciplinary Digital Publishing Institute.

[22] J. Pearl and D. Mackenzie, *The Book of Why: The New Science of Cause and Effect*. USA: Basic Books, Inc., 2018.

[23] Y. Li, K. B. Korb, and L. Allison, “Markov Blanket Discovery using Minimum Message Length,” *arxiv*, July 2021. arXiv: 2107.08140.

[24] Z. Ling, K. Yu, Y. Zhang, L. Liu, and J. Li, “Causal Learner: A Toolbox for Causal Structure and Markov Blanket Learning,” Mar. 2021. arXiv: 2103.06544.

[25] Y. Lu, A. Meisami, and A. Tewari, “Efficient Reinforcement Learning with Prior Causal Knowledge,” in *Proceedings of the First Conference on Causal Learning and Reasoning*, pp. 526–541, PMLR, June 2022. ISSN: 2640-3498.

[26] N. Sajid, P. J. Ball, T. Parr, and K. J. Friston, “Active Inference: Demystified and Compared,” *Neural Computation*, vol. 33, pp. 674–712, Mar. 2021. Publisher: MIT Press.

[27] R. Smith, K. J. Friston, and C. J. Whyte, “A step-by-step tutorial on active inference and its application to empirical data,” *Journal of Mathematical Psychology*, vol. 107, p. 102632, 2022.

[28] M. Gasse, D. Grasset, G. Gaudron, and P.-Y. Oudeyer, “Causal Reinforcement Learning using Observational and Interventional Data,” June 2021. arXiv:2106.14421 [cs].