European data space
for smart communities

Deliverable 3.2

DS4SSCC Deliverable 3.2 Timelapse viewing slider for 2D, 3D and Unreal viewing validated

WP 3 – Model usage phase

Authors: Corné Helmons (Rotterdam)
Reviewers: Niek Verschure ('s-Hertogenbosch), Anni Joela (Tampere), Margriet Visscher (Future Insight)
Delivery date: 19 June 2026
Dissemination level: Sensitive
Type: Report

Funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or of the granting authority. Neither the European Union nor the granting authority can be held responsible for them

Revision History

Author Name, Partner short name	Description	Date
Corné Helmons, Rotterdam	City Project Manager	19-June-2025

Abbreviations

Abbreviation	Definition	Abbreviation	Definition
DS4SSCC	European data space for smart communities (DS4SSCC)	DT-aCoGo	Digital Twinning as Common Good
OASC	Open & Agile Smart Cities & Communities	OUP	Open Urban Platform
WP	Work Package	PC	Pilot Coordinator
BIM	Building Information Modeling	WFS	Web Feature Service
OGC	Open Geospatial Consortium	ETL	Extract, Transform, Load
LOD	Level of Detail	UMG	Unreal Motion Graphics
REST	Representational State Transfer		

Table of contents

1. Executive Summary 4
2. Introduction 5
3. Problem Definition 6
4. Investigated Methodologies 7
4.1 Introduction 7
4.2 Masking (Visual-Based approach) 7
4.3 Filtering (Lifecycle-Based approach) 8
4.4 Comparison and design implications 9
4.5 Design decision 9
5. User interface and interaction concept 10
5.1 Introduction 10
5.2 Widget concept and key features 10
5.3 Technical interaction flow (Blueprint implementation) 11
5.4 User interaction scenario 14
5.5 Design considerations 15
5.6 Conclusion 15
6. Architectural integration and reusability 16
6.1 Introduction 16
6.2 Position within the DT-aCoGo architecture 16
6.3 Role of lifecycle attributes 17
6.4 Distributed filtering across applications 17
6.5 Integration with analytics models 17
6.6 Interoperability across applications 18
6.7 Reusability and sharing 18
6.8 Conclusion 18
7. Final conclusion and next steps 19
7.1 Final conclusion 19
7.2 Key outcomes 20
7.3 Next steps 20
7.4 Final reflection 21
8. Appendix — Notes on Standards & Interoperability 22
9. Retrospective 23

1. Executive Summary

This deliverable presents the design, implementation, and evaluation of a timelapse slider functionality within Unreal Engine as part of the DT-aCoGo digital twin framework. The work builds directly upon the principles established in Deliverable 2.2, in which time is defined as an intrinsic property of urban data rather than a purely visual feature.

The primary objective of the timelapse slider is to enable users to explore urban environments across different moments in time, supporting both visualisation and analytical use cases. In the context of urban development, datasets representing the current state (e.g. 3D Tiles) and future developments (e.g. BIM models) inherently overlap. Without a consistent temporal mechanism, this leads to visual clutter, incorrect analytical results, and limited decision support.

The key design principle adopted in this work is the use of lifecycle-based filtering, where objects are enriched with temporal attributes (ValidFrom, ValidTo) during data preparation. This approach ensures that all applications interpret time consistently by applying the same simple filtering rule based on a selected time parameter.

Within Unreal Engine, this principle is implemented through a Blueprint-based timelapse slider widget, which acts as a user interface control for selecting a moment in time. The widget dynamically determines the temporal range based on available data, evaluates lifecycle attributes at runtime, and includes or excludes objects accordingly. Unlike visual masking techniques, this approach ensures that objects are not merely hidden, but structurally filtered out of the scene, avoiding issues with collisions, performance, and analytical inconsistencies.

The analysis presented in this deliverable demonstrates that visual masking approaches are insufficient, as they operate only at the rendering level and do not align with the data-driven architecture. In contrast, lifecycle-based filtering provides a robust, scalable and interoperable solution that can be consistently applied across visualisation environments (e.g. Unreal Engine, web viewers) and analytical services.

A key added value of the timelapse slider is its ability to ensure that analytical results are presented within the correct temporal context. By linking analysis models to the same time parameter, results such as heat stress simulations can be shown only when relevant to the selected moment. This prevents misinterpretation, for example by avoiding the display of future analysis results in the context of the current situation, and significantly improves the clarity and reliability of decision support.
The implementation in Unreal Engine confirms that lifecycle-based filtering can be realised in a real-time 3D environment through custom metadata parsing and runtime evaluation, despite limited native support for temporal semantics. The result is a reference implementation that demonstrates how the same mechanism can be applied consistently across different applications without modifying the underlying data.

In conclusion, the Unreal Engine timelapse slider is not a standalone feature, but a key component of a broader architectural approach in which time is embedded in the data and interpreted consistently across systems. This enables scalable reuse across cities and applications, supports interoperability within the DT-aCoGo ecosystem, and strengthens the integration between visualisation and analytical workflows.

2. Introduction

This document presents the implementation of the timelapse viewing slider within Unreal Engine for the DT-aCoGo pilot. It builds directly upon earlier work in which the conceptual and architectural foundations for time-based functionality in the digital twin were defined.
In the preceding phase of the project, a dedicated deliverable (D2.2) analysed the challenges related to representing urban change over time within a digital twin environment. That analysis demonstrated that:

- urban datasets inherently overlap in time, combining existing situations with future developments;
- purely visual solutions lead to inconsistencies between what users see and what analytical models process;
- and a scalable solution requires a data-driven approach rather than a viewer-specific implementation.

As a result, D2.2 established a key design principle:

Time must be treated as an intrinsic property of the data, implemented through lifecycle attributes such as ValidFrom and ValidTo, and interpreted consistently across all applications.

This implementation report should therefore be read as a direct continuation and practical realisation of that principle, focusing specifically on its application within Unreal Engine.

The objective of this document is to:

- translate the previously defined lifecycle-based concept into a working Unreal Engine implementation,
- evaluate engine-specific solution strategies,
- and document the design decisions and trade-offs required to achieve consistent temporal filtering in a real-time 3D environment.

By doing so, this report bridges the gap between conceptual architecture (D2.2), as shown in Figure 1, and practical implementation, ensuring that the chosen approach remains aligned with the broader DT-aCoGo principles of consistency, scalability, and interoperability.

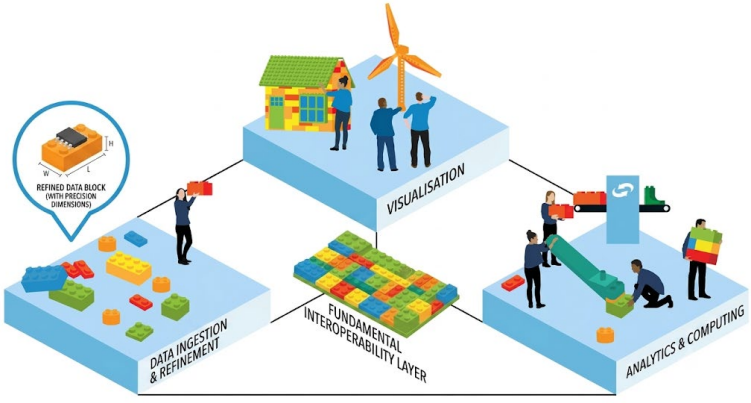


Figure 1: Schematic architecture DT-aCoGo

3. Problem Definition

Within the broader DT-aCoGo architecture, multiple applications—such as web-based viewers (e.g. CesiumJS), analytical services (e.g. WFS/GeoJSON), and game engine environments like Unreal Engine—consume the same harmonised datasets via the Open Urban Platform (OUP) and the Purple Polar Bear (PPB) common model layer. In this federated setup, it is essential that all applications rely on the same temporal logic, ensuring that visualisation and analysis remain aligned.

The core problem addressed in this deliverable is therefore not simply how to implement a timelapse slider in Unreal Engine, but how to ensure that this implementation strictly adheres to the lifecycle-based filtering logic defined in Deliverable 2.2.
If Unreal Engine were to implement its own, engine-specific solution—such as purely visual masking or custom interaction logic that does not rely on lifecycle attributes—this would introduce deviations from the shared data-driven approach. As a consequence:

- Visualisations in Unreal would no longer correspond to the results of analytical services;
- Different visualisation platforms would interpret the same datasets differently;
- The scalability and reusability of the solution across cities and applications would be compromised.

This directly conflicts with the DT-aCoGo objective of establishing a single, reusable and application-independent mechanism for time-based filtering, enabling consistent use across multiple domains and tools.

The problem can therefore be formulated as:

How can the timelapse slider in Unreal Engine be implemented in such a way that it fully reuses the lifecycle-based filtering mechanism defined in Deliverable 2.2, ensuring that the same logic can be applied consistently across other visualisation and analytical applications?

This problem definition leads to several key requirements for the Unreal implementation:

- The timelapse slider must function solely as a provider of a time parameter, without modifying datasets or geometry;
- All filtering must be based on lifecycle attributes (ValidFrom, ValidTo) prepared in the data layer via PPB;
- The filtering rule applied in Unreal must be identical to the rule used in other applications, ensuring consistency between visualisation and analysis;
- The implementation should act as a reference implementation, demonstrating how the same mechanism can be applied in other environments beyond Unreal.

By positioning the Unreal timelapse slider in this way, the solution contributes not only to the specific use case of immersive visualisation, but also to the broader goal of creating a generic, scalable and interoperable temporal filtering mechanism across the DT-aCoGo ecosystem.

4. Investigated Methodologies

4.1 Introduction

To implement a timelapse slider within Unreal Engine, multiple solution approaches were evaluated. The objective was not only to achieve a correct visual representation of time-dependent urban scenarios, but to ensure alignment with the lifecycle-based approach defined in Deliverable 2.2.

A key requirement is that the selected solution must remain consistent with the overarching architectural principle that temporal behaviour is driven by data (lifecycle attributes), and interpreted uniformly across applications.

Within this context, two main implementation strategies emerged:

- Masking (visual-based approaches)
- Filtering (data-driven approaches based on lifecycle attributes)

4.2 Masking (Visual-Based approach)

Masking refers to techniques where objects are not actually removed or excluded from the dataset, but are visually hidden or made transparent within the rendering environment. In Unreal Engine, this can be achieved through material manipulation (e.g. opacity), clipping, or shader-based effects.

Characteristics

- Objects remain present in the scene and in the underlying data structures
- Visibility is controlled purely at the rendering level
- Typical implementations include:
    - Transparency or opacity masking
    - Clipping geometries
    - Shader-based visual filtering

Advantages

- Relatively easy to implement in a game engine environment
- Immediate visual feedback for prototyping
- No modification required to the data structure

Limitations

- No impact on underlying data: objects that are visually hidden are still part of the scene graph and data queries
- Inconsistency with analytical applications: analytical services (e.g. WFS, GeoJSON) still process all objects, including those visually masked
- Collision and interaction issues: in Unreal, masked objects may still participate in physics, collisions, and interaction logic
- Performance overhead: rendering hidden objects can still incur computational cost
- Non-reusable logic: masking is specific to the rendering engine and cannot be easily replicated in other environments

Assessment
While masking can produce visually acceptable results, it does not align with the lifecycle-based approach defined in Deliverable 2.2, where time is an intrinsic property of the data and must be interpreted consistently across systems.

As such, masking is considered suitable for prototyping or purely visual demonstrations, but not for a scalable and interoperable solution.

4.3 Filtering (Lifecycle-Based approach)

Filtering refers to the inclusion or exclusion of objects based on temporal attributes, specifically lifecycle attributes such as ValidFrom and ValidTo, as defined in Deliverable 2.2.

In this approach, objects are evaluated against a selected time parameter (provided by the timelapse slider), and are either included or excluded from the visualisation and processing pipeline.

Characteristics

- Objects are conditionally included based on lifecycle attributes
- Filtering is applied consistently across visualisation and analytical environments
- The same rule is used across applications:

An object is visible if: ValidFrom ≤ selected_time < ValidTo

Advantages

- Consistency across applications: the same logic applies in Unreal, web viewers, and analytical services
- Alignment with data-driven architecture: filtering operates on lifecycle attributes prepared in the data layer (PPB)
- No side effects in simulation or interaction: excluded objects are not part of the active scene
- Scalability: supports multiple overlapping projects and dynamic timelines
- Reusability: the same mechanism can be implemented in other engines (e.g. CesiumJS) without modification of the data

Limitations

- Requires access to lifecycle attributes within the Unreal environment
- May require custom parsing or metadata handling, depending on how data is ingested (e.g. 3D Tiles in Unreal)
- Slightly higher implementation complexity compared to basic visual masking

Assessment
Filtering based on lifecycle attributes is the only approach that satisfies the architectural requirements defined in Deliverable 2.2, ensuring:

- a single source of truth
- consistency between visualisation and analysis
- reuse across multiple applications and cities

4.4 Comparison and design implications

Aspect	Masking	Filtering
Level of implementation	Visual layer only	Data-driven + application logic
Impact on data	None	Determines inclusion/exclusion
Consistency with analytics	No	Yes
Behaviour in Unreal (collisions, physics)	Still active	Correctly excluded
Reusability across applications	Low	High
Alignment with D2.2	No	Yes

4.5 Design decision

Based on these findings, the filtering approach is selected as the foundation for the Unreal Engine timelapse slider implementation.

This decision implies that:

- The timelapse slider in Unreal Engine functions solely as a provider of a time parameter;
- All temporal behaviour is derived from lifecycle attributes embedded in the data;
- Objects are explicitly included or excluded from the scene based on the shared filtering rule (ValidFrom ≤ selected_time < ValidTo);
- The implementation must rely on data parsing and evaluation logic within Unreal, rather than visual techniques such as material-based masking.

In practice, this requires the use of a custom parsing mechanism within Unreal Engine to interpret lifecycle attributes and apply filtering at runtime. While web-based viewers (e.g. CesiumJS) may provide more direct support for metadata-driven filtering, Unreal Engine requires additional implementation effort to achieve the same behaviour.

Implications

By selecting filtering over masking, the Unreal implementation becomes:

- Consistent with analytical services and other visualisation platforms;
- Scalable, supporting multiple datasets and overlapping development scenarios;
- Reusable, as the same logic can be implemented across different engines and applications without modifying the underlying data;
- Aligned with DT-aCoGo architecture, where time is treated as a property of the data rather than a visual effect.

Final Conclusion

The Unreal Engine timelapse slider should therefore not be implemented as a standalone visual feature, but as a reference implementation of the lifecycle-based filtering mechanism defined in Deliverable 2.2.
This ensures that the same temporal logic can be applied consistently across all visualisation and analytical applications within the DT-aCoGo ecosystem, enabling a unified and interoperable approach to time-based urban data exploration.

5. User interface and interaction concept

5.1 Introduction

The user interface of the Unreal Engine timelapse slider is designed as a lightweight interaction layer built on top of the lifecycle-based filtering mechanism defined in Deliverable 2.2.

In line with the DT-aCoGo architecture, the UI does not introduce new temporal logic. Instead, it provides a user-friendly control mechanism for a single time parameter, which is interpreted by the application using lifecycle attributes (ValidFrom, ValidTo

The implementation in Unreal Engine demonstrates how this generic mechanism can be realised within a real-time 3D environment, while remaining fully consistent with other applications in the ecosystem.

5.2 Widget concept and key features

The timelapse slider is implemented as a standalone UI widget, fully developed using Unreal Engine's Blueprint system. This ensures accessibility, flexibility, and ease of integration without requiring C++ development.

Key characteristics of the widget include:

- **100% Blueprint-based implementation**
  The entire logic, from UI construction to filtering behaviour, is implemented using Unreal's visual scripting system, making it transparent and extensible.
- **Dynamic range generation**
  The slider does not rely on predefined time ranges. Instead, it automatically determines the minimum and maximum dates present in the dataset by scanning lifecycle attributes within the scene.
- **Lifecycle-based attribute filtering**
  The widget directly interfaces with object metadata (ValidFrom, ValidTo), enabling consistent filtering behaviour aligned with Deliverable 2.2.
- **Adjustable temporal resolution**
  The slider supports configurable intervals (e.g. yearly steps), allowing adaptation to different use cases such as planning phases or long-term urban development.

These features ensure that the UI remains data-driven rather than hardcoded, supporting reuse across datasets and applications.

5.3 Technical interaction flow (Blueprint implementation)

The internal working of the timelapse slider is implemented through Unreal Engine's **EventGraph**, which defines the runtime behaviour of the widget.

Event Construct

The timelapse slider is activated via the main application menu, managed by a central state manager. When activated, the widget is constructed and the following processes are triggered:

- Initialization of the UI
- Retrieval of dataset information
- Binding to application-level update events

This ensures that the timelapse slider is always synchronised with the current state of the application.

Update Date Limits

This function determines the temporal range of the slider by extracting lifecycle attributes from the scene:

1. All Cesium 3DTiles actors in the scene are accessed
2. Relevant components are identified (e.g. CesiumGltfInstancedComponent, CesiumGltfPrimitiveComponent)
3. Metadata is extracted from the property tables
4. Lifecycle attributes (ValidFrom, ValidTo) are retrieved
5. The global minimum and maximum dates are computed

This step ensures that the UI dynamically adapts to the data available in the scene, without manual configuration.

Construct Slider Items

Based on the derived minimum and maximum dates, the timelapse slider is configured:

- The temporal range is established
- Slider values are generated based on a defined interval (e.g. per year)
- UI elements are aligned to represent discrete or continuous time steps

This results in a timeline that accurately reflects the temporal extent of the dataset.

Data Synchronisation (Event Binding)

The timelapse slider listens to updates in the dataset through an event dispatcher mechanism:

- When datasets are added or removed, the widget is notified
- The temporal range and slider configuration are automatically recalculated

This ensures that the UI remains consistent with dynamically changing data sources. An impression of this mechanism defined as a blueprint is given in figure 2.

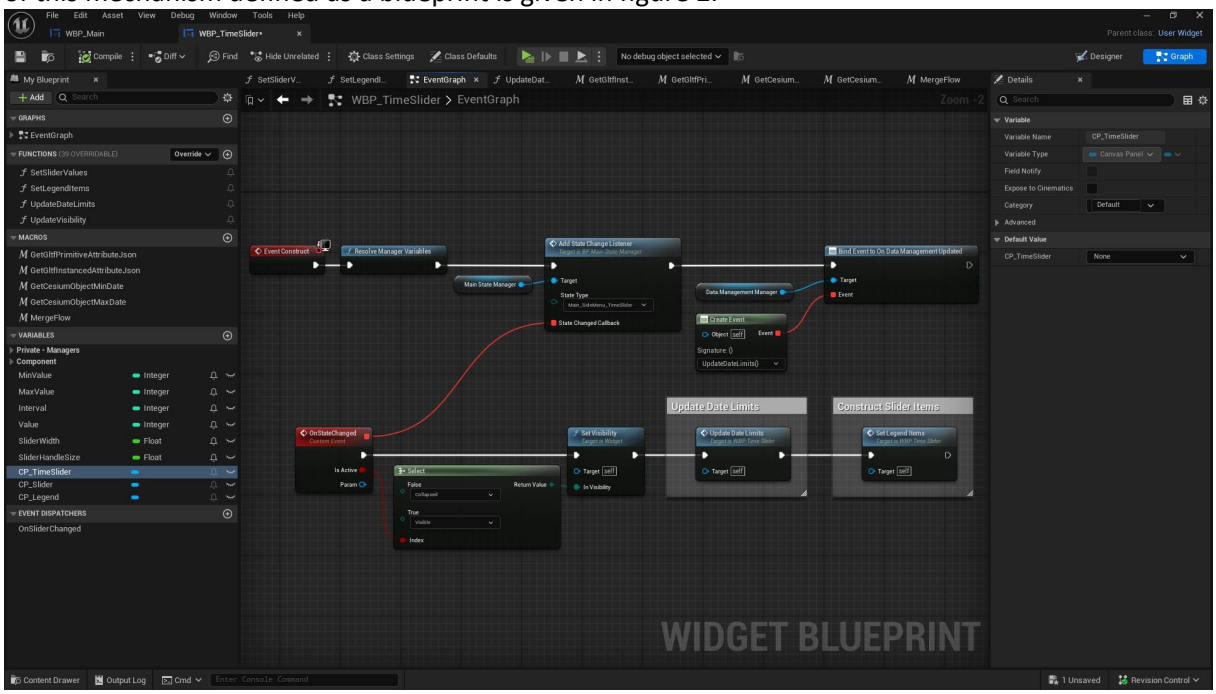


Figure 2: Blueprint for dynamic timelapse slider

OnSliderChanged Event

The core interaction is handled through the slider change event:

- The user selects a specific moment in time
- The selected value is stored as the global time parameter
- For each object in the scene:
  - Lifecycle attributes are evaluated
  - The condition ValidFrom ≤ selected_time < ValidTo is applied
- Objects are either:
  - made visible (included), or
  - excluded from rendering

An impression of the mechanism for filtering objects defined as a blueprint is given in figure 3.

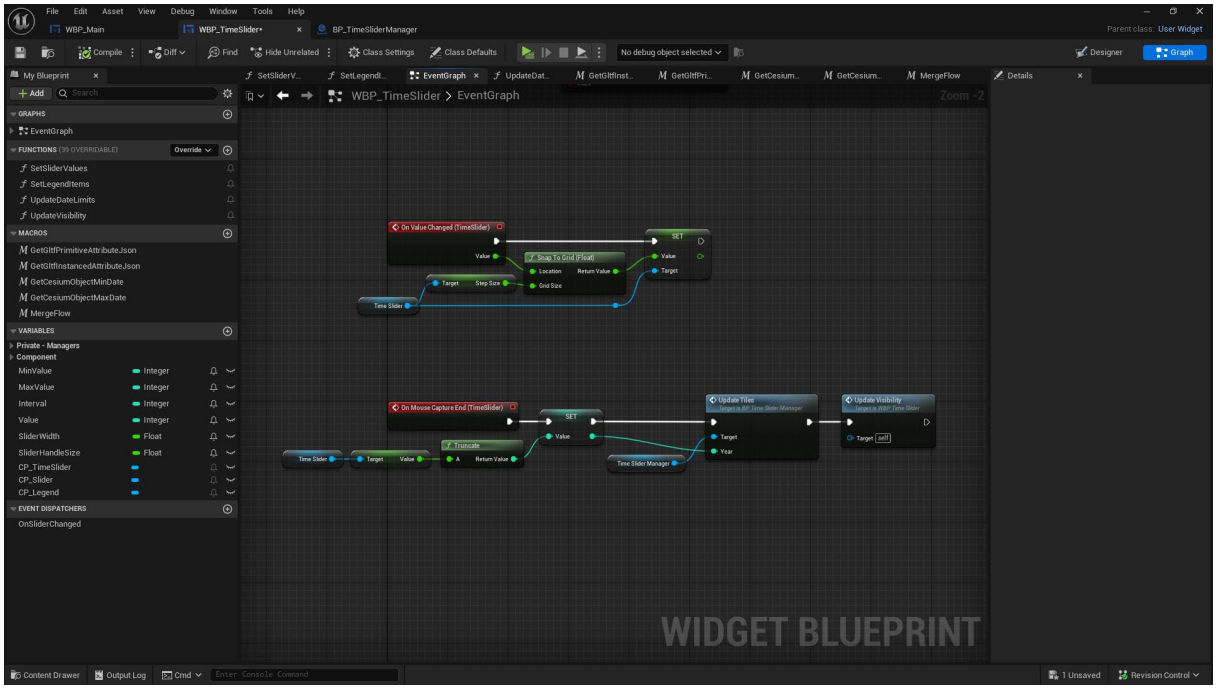


Figure 3: Blueprint for timelapse slider functionality

This process implements the lifecycle-based filtering principle directly within the Unreal runtime.

5.4 User interaction scenario

A typical interaction scenario demonstrates the effect of the timelapse slider:

- At the initial time (current state):
  - Existing city models (e.g. 3D Tiles) are visible
  - Future BIM models are not yet active
- As the user moves the slider forward in time:
  - Objects whose lifecycle has ended disappear
  - New objects become visible as their ValidFrom condition is met
- At a future point in time:
  - The original situation is no longer visible in the project area
  - The new BIM-based development fully appears

This results in a clear transition where existing conditions are replaced by planned developments, entirely driven by data filtering rather than visual effects. An impression of the user interface for the timelapse slider is given in figure 4 below.



Figure 4: Userinterface of timelapse slider

5.5 Design considerations

The chosen implementation approach has several important implications:

- **No reliance on visual masking**
  Objects are not made transparent, but structurally excluded, avoiding issues with collisions and interactions
- **Custom parsing of metadata required**
  Unlike some web-based solutions, Unreal requires explicit handling of metadata (e.g. from Cesium 3D Tiles), implemented via Blueprint logic
- **Consistent behaviour across data layers**
  All datasets respond to the same time parameter, ensuring a unified view
- **Extensibility through Blueprint**
  The implementation can be extended or shared without deep engine modifications

5.6 Conclusion

The Unreal Engine timelapse slider UI demonstrates how a data-driven temporal filtering mechanism can be translated into an intuitive user interaction model.
By combining:

- dynamic UI generation,
- lifecycle-based filtering, and
- Blueprint-based implementation,

the solution provides a practical and reusable implementation of the principles defined in Deliverable 2.2.

Importantly, the Unreal implementation does not introduce application-specific behaviour, but serves as a reference implementation showing how the same temporal logic can be applied in other visualisation environments within the DT-aCoGo ecosystem.

6. Architectural integration and reusability

6.1 Introduction

Following the design decision to adopt lifecycle-based filtering (Chapter 3) and its concrete implementation within Unreal Engine (Chapter 5), this chapter describes how the timelapse slider solution fits within the broader DT-aCoGo architecture.

The key objective is not only to realise a functional implementation in Unreal Engine, but to ensure that this implementation remains fully aligned with the data-driven principles defined in Deliverable 2.2, enabling reuse across applications, cities, and use cases.

6.2 Position within the DT-aCoGo architecture

The DT-aCoGo architecture is based on a clear separation between data, processing, and application layers, as shown in figure 5.

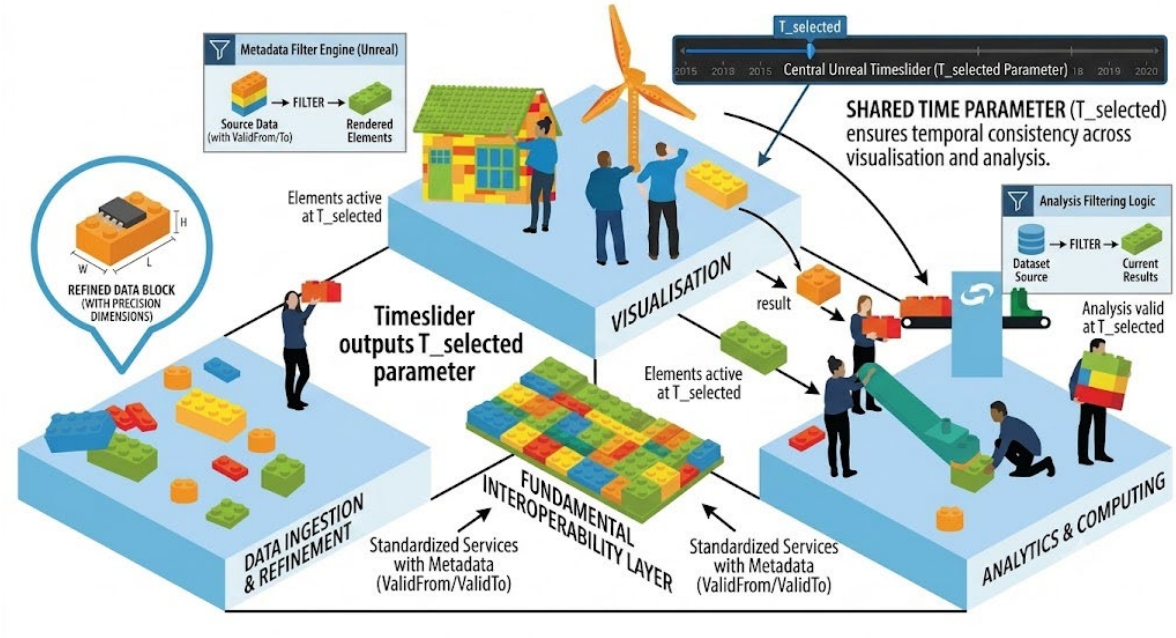


Figure 5: Conceptual architecture of timelapse slider

Within this structure:

- Data services (e.g. 3D Tiles, WFS, GeoJSON) are made available through the Open Urban Platform (OUP);
- The Purple Polar Bear (PPB) layer harmonises and enriches datasets with lifecycle attributes;
- Applications (such as Unreal Engine, web viewers, and analytical tools) consume these datasets and apply filtering logic locally.

The Unreal timelapse slider operates at the application layer, where it interprets the shared temporal structure embedded in the data. This ensures that all applications operate on the same principles, without introducing application-specific deviations.

6.3 Role of lifecycle attributes

Lifecycle attributes (ValidFrom, ValidTo) form the foundation for temporal interpretation within the DT-aCoGo ecosystem.
These attributes:

- Define the temporal validity of each object;
- Enable non-destructive filtering of datasets;
- Act as a shared semantic contract between data and applications.

By relying exclusively on these attributes, the Unreal implementation guarantees that temporal behaviour is:

- consistent across visualisation environments,
- aligned with analytical services,
- and independent of the underlying technology stack.

6.4 Distributed filtering across applications

A defining characteristic of the architecture is that filtering is implemented in a distributed manner across all consuming applications.
This implies that:

- The timelapse slider provides a shared time parameter;
- Each application applies the same filtering rule (ValidFrom ≤ selected_time < ValidTo);
- No centralised filtering service is required.

This approach ensures:

- consistency between visualisation and analytical results;
- flexibility in application design;
- scalability across multiple environments and cities.

The Unreal implementation demonstrates how this distributed filtering principle can be realised in a real-time 3D engine through Blueprint-based logic and metadata parsing.

6.5 Integration with analytics models

An important implication of the lifecycle-based approach is that it does not only apply to visualisation, but also to analytical models and their results.
Because both input data and visualisation are governed by the same time parameter, analytical processes can be aligned with the selected moment in time. This ensures that analysis results are always interpreted within the correct temporal context.
For example:

- A heat stress analysis calculated for a future scenario should only be visible when the timelapse slider is set to that future state;
- It should not be displayed in the context of the current situation, as this would lead to misinterpretation.

By linking analytical outputs to the same lifecycle-based filtering mechanism:

- analysis results can be dynamically shown or hidden based on the selected time;
- consistency is ensured between data, visualisation, and analysis;
- users are provided with clear and unambiguous insights.

This integration prevents situations where results from different temporal contexts are mixed, which could otherwise lead to incorrect conclusions or decision-making.

6.6 Interoperability across applications

Because the solution is based on lifecycle attributes and a simple evaluation rule, it is inherently technology-independent.
This allows the same mechanism to be applied in:

- Unreal Engine (via Blueprint-based filtering)
- Web-based viewers such as CesiumJS
- Analytical platforms consuming WFS or GeoJSON services

The Unreal implementation demonstrates that even in environments where metadata handling requires additional effort, the same architecture can be preserved without compromise.

6.7 Reusability and sharing

The Unreal timelapse slider is implemented as a modular Blueprint component, supporting reuse across projects:

- It can be integrated into other Unreal environments;
- The logic is transparent and extendable;
- It can serve as a reference implementation for other partners.

An important consideration is the potential to share this implementation more broadly, for example as:

- a reusable reference within the DT-aCoGo ecosystem;
- a shared component across participating cities;
- or a contribution to the wider community

Such reuse would accelerate adoption and ensure consistent implementation of lifecycle-based temporal filtering.

6.8 Conclusion

The Unreal Engine timelapse slider is fully integrated within the DT-aCoGo architecture as an application-layer implementation of a shared temporal concept.
By combining lifecycle-based data enrichment with distributed filtering and a unified time parameter, the solution ensures:

- consistency between visualisation, data, and analytical results;
- interoperability across applications and cities;
- clarity for users by presenting information within the correct temporal context;
- and prevention of misinterpretation by avoiding the mixing of current and future scenarios.

As a result, the timelapse slider not only improves visual interaction, but also strengthens the interpretability and reliability of analytical outcomes, making it a key component in data-driven urban decision-making.

7. Final conclusion and next steps

7.1 Final conclusion

This deliverable demonstrates that the implementation of a timelapse slider in Unreal Engine can only be considered successful when it is fully aligned with the lifecycle-based filtering approach defined in Deliverable 2.2.
The core insight is that time must not be implemented as a visual effect, but as a fundamental property of the data, expressed through lifecycle attributes such as ValidFrom and ValidTo. By adhering to this principle, the Unreal Engine implementation becomes part of a broader, consistent system rather than a standalone solution.

The analysis of masking versus filtering clearly shows that:

- Visual masking approaches are insufficient, as they only affect rendering and introduce inconsistencies with analytical results;
- Lifecycle-based filtering provides a robust, scalable and interoperable solution, ensuring that both visualisation and analysis operate on the same temporal logic.

The Unreal Engine implementation confirms that this approach is technically feasible, even in environments where metadata handling requires custom logic. Through a Blueprint-based implementation, lifecycle attributes can be parsed and evaluated at runtime, enabling consistent behaviour in line with the overall architecture.

In addition, the integration of the timelapse slider with analytical models highlights an essential benefit of the approach:

- Temporal filtering ensures that analysis results are always presented in the correct time context;
- It prevents situations where results from future scenarios (e.g. heat stress simulations) are interpreted within the current situation;
- It creates a coherent link between data, visualisation, and analysis, improving interpretability and decision support.

As a result, the timelapse slider is not merely a UI component, but a critical mechanism for ensuring consistency and reliability in digital twin environments.

7.2 Key outcomes

The main outcomes of this work can be summarised as follows:

- A validated implementation approach for a timelapse slider in Unreal Engine based on lifecycle attributes
- Demonstration that distributed filtering across applications ensures consistency between visualisation and analytics
- A Blueprint-based reference implementation, showing how lifecycle-based filtering can be realised in a real-time 3D engine
- Confirmation that the same mechanism can be applied across multiple environments (e.g. Unreal Engine, web viewers, analytical platforms)
- Improved clarity and reliability for end users, by ensuring that all outputs are shown in the correct temporal context

These outcomes contribute directly to the DT-aCoGo objective of creating a scalable, interoperable and reusable digital twin ecosystem.

7.3 Next steps

While the current implementation provides a solid foundation, several next steps are identified to further develop and operationalise the solution:

1. Standardisation of Temporal Filtering in 3D Environments
Further work is needed to:
    - Align lifecycle-based filtering with emerging standards (e.g. OGC / 3D Tiles evolution);
    - Ensure that temporal metadata can be natively supported across more platforms;
    - Reduce the need for custom parsing in environments like Unreal Engine.

This aligns with ongoing discussions about extending temporal capabilities in 3D data services.

2. Reusability and Sharing of the Unreal Implementation
The Blueprint-based implementation can serve as a reference component for other stakeholders:
    - Evaluate whether the Blueprint logic can be shared further then the DT-aCoGo consortium;
    - Explore the possibility of publishing it as a reusable module or open-source component;
    - Document the implementation to support adoption by other cities and developers.

Sharing the implementation would accelerate uptake and promote consistent usage of lifecycle-based filtering.

3. Integration with Analytical Workflows
The link between timelapse slider and analytical models should be further operationalised:
    - Ensure that analytical services (that use e.g. WFS / GeoJSON) explicitly support time-based filtering;
    - Align simulation outputs with lifecycle attributes or equivalent temporal metadata;
    - Develop workflows where analyses are automatically recalculated or filtered based on the selected time.

This will strengthen the role of the timelapse slider as a central coordination mechanism between visualisation and analysis.

4. Enhanced User Interaction and Scenario Management
Future iterations of the UI may include:

- Scenario-based navigation (e.g. switching between alternative development plans);
- Playback functionality (animated progression through time);
- Integration of key milestones or events into the timeline;
- Improved feedback mechanisms for users (e.g. highlighting changes).

These enhancements can improve usability while maintaining the underlying data-driven principles.

5. Scaling Across Cities and Use Cases
To support the DS4SSCC objectives, the solution should be validated in additional contexts:

- Apply the timelapse slider to datasets from multiple cities based on the replicable recipes;
- Connect it with different types plug-ins for a variety of use cases (e.g. mobility, climate, infrastructure);
- Evaluate performance and usability at city scale.

This will confirm the scalability and general applicability of the approach.

7.4 Final reflection

The implementation of the Unreal Engine timelapse slider demonstrates that temporal functionality in digital twins should be designed as a shared, data-driven mechanism rather than as an application-specific feature.

By combining lifecycle-based data enrichment, distributed filtering, and intuitive user interaction, the project establishes a robust foundation for time-aware digital twins that can be reused across applications, domains, and cities.

This approach not only improves visualisation, but also ensures that analytical insights are consistently interpreted in the correct temporal context—ultimately supporting better-informed and more reliable urban decision-making.

8. Appendix — Notes on Standards & Interoperability

An important outcome of this work is that the proposed approach is not only applied within the project context, but is also intended to contribute to the further development of open standards. The lifecycle-based timelapse functionality will be brought forward within the Open Geospatial Consortium (OGC), specifically in relation to the evolution of the 3D Tiles specification towards version 2.0.

While temporal attributes and filtering mechanisms are already supported in standards such as:

- OGC Web Feature Service (WFS), and
- emerging OGC API standards,

equivalent functionality is not yet uniformly embedded in 3D Tiles in a way that ensures full interoperability across platforms.

By proposing the integration of lifecycle semantics (ValidFrom, ValidTo) into the 3D Tiles ecosystem, this work contributes to:

- aligning 3D visualisation standards with existing OGC temporal capabilities,
- enabling application-independent time filtering,
- supporting consistent behaviour across 2D and 3D data services,
- and promoting standardised, reusable implementations across different engines and platforms.

This step is essential to ensure that time-based functionality in digital twins becomes a standard capability, rather than a project-specific custom solution.

9. Retrospective

The Unreal Engine implementation demonstrates that advanced visualisation environments can successfully adopt a data-driven temporal model, even when native support is limited. However, the real challenge lies in ensuring that this implementation is not isolated, but forms part of a coherent and interoperable ecosystem.

By enforcing the same lifecycle-based logic across visualisation and analytical platforms, the project establishes a consistent and reliable approach to time-based digital twin functionality.

This retrospective highlights that the success of the timelapse slider is not defined by its technical implementation in Unreal, but by its ability to function as a shared mechanism across all applications, ensuring that users always interact with data and analyses in the correct temporal context.

About the European Data Space for Smart Communities

The European Data Space for Smart Communities (DS4SSCC-DEP) initiative is a pivotal deployment following the preparatory action for a Data Space for Sustainable and Smart Cities and Communities (DS4SSCC). Emphasising sustainability aspects and diversity in communities, DS4SSCC developed a multi-stakeholder data governance scheme, created a blueprint for the European DS4SSCC, delivered priority datasets, and developed a roadmap towards a mature DS4SSCC.

DS4SSCC-DEP's vision revolves around creating a well-governed federated territorial, place-based data space for smart communities available for developers and infrastructure providers, aligning with the prospects outlined in Europe's Digital Decade objectives. This approach distinguishes itself from sectoral data spaces by encompassing diverse domains, and underpinning governance across all levels of society. The key objectives of the project are:

- **Objective 1: Establishing a Federated and Innovative Data Space**. This phase focuses on creating a large-scale data space controlled by public data holders, ensuring alignment with the Smart Middleware Platform and broader data space ecosystems. It also aims to offer middleware service solutions facilitating data sharing and management while refining the blueprint based on gained experience.
- **Objective 2: Pilots and Refinement of the Data Space Blueprint**. Through 10-12 cross-sectoral data pilots in various EU communities, this objective aims to validate and refine the data space blueprint. Using common high-value datasets, it seeks to create added value by combining data from various domains, contributing to the sustainability plan, and minimising infrastructure investment.
- **Objective 3: Fostering Innovation and Ecosystem Alignment**. Engaging stakeholders, compliance with sector-specific legislation, establishing links with Horizon Europe missions, creating services available via trusted application catalogues, and enhancing ethical considerations and AI-enabled local solutions constitute the focus of this objective.

DS4SSCC-DEP, building upon established European networks, aims to significantly contribute to the sustainability goals of European citizens. It emphasises co-creation, standardisation, business models, strategies for data spaces, and envisions a federated platform.

DS4SSCC-DEP consortium includes the following partners: