UNDERWORLDS: Cascading Situation Assessment for Robots

Séverin Lemaignan[1], Yoan Sallami[2], Christopher Wallbridge[3],
Aurélie Clodic[2], Tony Belpaeme[3], and Rachid Alami[2]

Abstract— **We introduce UNDERWORLDS, a novel lightweight framework for *cascading spatio-temporal situation assessment* in robotics. UNDERWORLDS allows programmers to represent the robot's environment as real-time distributed data structures, containing both scene graphs (for representation of 3D geometries) and timelines (for representation of temporal events). UNDERWORLDS supports *cascading* representations: the environment is viewed as a set of *worlds* that can each have different spatial and temporal granularities, and may inherit from each other. UNDERWORLDS also provides a set of high-level client libraries and tools to introspect and manipulate the environment models.**

This article presents the design and architecture of this open-source tool, and explores some applications, along with examples of use.

I. INTRODUCTION

UNDERWORLDS is a distributed and lightweight open-source framework[1] that enables robot programmers to build and refine spatial and temporal models of the environment surrounding a robot in real-time. UNDERWORLDS makes it possible to share these world models amongst the software components running on the robot. Additionally, UNDERWORLDS enables users to represent and manipulate *multiple alternatives* to the current, perceived world model in a distributed manner. For instance, the world with some objects filtered out; the world 'viewed' from the perspective of another agent; a hypothetical world resulting from the simulated application of a plan, etc.

A. Distributed Situation Assessment

Anchoring perceptions in a symbolic model suitable for decision-making requires perception abilities and their symbolic interpretation. We call *physical situation assessment* the cognitive skill that a robot exhibits when it represents and assesses the nature and content of its surroundings and monitors its evolution.

Numerous approaches exist, like amodal (in the sense of modality-independent) *proxies* [1], grounded amodal representations [2], semantic maps [3], [4], [5] or affordance-based planning and object classification [6], [7].

UNDERWORLDS is specifically inspired by geometric and temporal reasoners like SPARK (*SPAtial Reasoning & Knowledge*) [8] or TOASTER (*Tracking Of Agents and Spatio-TEmporal Reasoning*) [9]. SPARK acts as a situation assessment reasoner that generates symbolic knowledge from the geometry of the environment with respect to relations between objects, robots and humans. It also takes into account the different perspective that each agent has on the environment. SPARK embeds a modality-independent geometric model of the environment that serves both as basis for the fusion of the perception modalities and as bridge with the symbolic layer [10]. This geometric model is built from 3D CAD models of the objects, furniture and robots, and full body, rigged models of humans. It is updated at run-time by the robot's sensors. Likewise, UNDERWORLDS embeds a grounded amodal model of the environment, updated online from the robot's sensors (sensor fusion).

However, SPARK is a monolithic module that does not support sharing its internal 3D model with other external components. In contrast, UNDERWORLDS focuses on offering a shared and distributed representation of the environment within the robot's software architecture. This also distinguishes UNDERWORLDS from complex cognitive toolkits like KnowRob (as found in OpenEASE [11]). While these tools maintain a spatio-temporal model of the world, this model is internal and not meant to be made widely accessible to other external processes. UNDERWORLDS focuses instead on reusability and sharing of distributed spatio-temporal models. As such, UNDERWORLDS can be seen as a middleware for spatio-temporal world models and, contrary to KnowRob, it does *not* provide any intrinsic high-level processing or reasoning capability. Such reasoning skills are implemented in loosely-coupled *clients* (see Section III hereafter).

Work on distributed scene graphs [12] has been previously applied to robotics to provide a shared 3D representation of the robot's environment (for instance, the *Robot Scene Graph* [13] or the *Deep State Representation* proposed in [14]). UNDERWORLDS offers a similar distribution mechanism for 3D scene graphs and extends it to temporal representations. Besides, UNDERWORLDS further extends this line of work by providing the ability to create, manipulate and share *multiple alternative worlds*. As an example, these could correspond to filtered or hypothetical *views* on the initial, perceived model of the environment.

B. Representing Alternative States of the World

The components which make use of spatial and temporal models of the environment are usually found in the intermediate layers of robotic architectures, between the low-level perceptual layers, and the high-level decisional layers. They include modules like geometric reasoners (that compute

---

[1]Author is with Bristol Robotics Lab, University of the West of England, Bristol, United Kingdom `severin.lemaignan@brl.ac.uk`, [2]Authors are with LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France `firstname.surname@laas.fr`, [3]Authors are with CRNS, Plymouth University, Plymouth, United Kingdom `firstname.surname@plymouth.ac.uk`

[1] `https://github.com/underworlds-robot/underworlds`

spatial and topological relations between objects), motion planners or action recognition modules.

These components exhibit different needs in terms of representation, like different nominal spatial and/or temporal resolutions. For instance, a 3D motion planner would typically use coarse 3D models of surrounding objects to lower the computational load while planning, while a module assessing the visibility of objects might need high-resolution models for accurate 3D visibility testing. This requirement of multiple task-specific representations has been framed as the need for *deep representations* by Beetz [15].

Traditional robotic middlewares, like ROS, are not particularly well suited to deal with these different needs: full geometric data can be represented, but is not first-class citizen: a basic task like displaying a 3D mesh at an arbitrary position is not particularly easy to perform with ROS, requiring the combination of static Collada meshes, a URDF kinematic description, TF broadcasters, and a 3D visualisation tool like RViz. Critically, simultaneously representing and reasoning on alternative states of the environment is not directly feasible.

Representing alternative states is however often highly desirable. For instance, software components manipulating environment models typically perform better if the models are physically consistent. However, low-level perception inaccuracies often introduce hard-to-avoid physical inconsistencies (like detected objects floating in the air, or wrongly inset into other objects). Therefore, a post-process stage (for instance, using a physics simulation engine) is needed to move the objects seen by the robot into physically-correct positions. Implemented with a classical approach (for instance, using ROS TF frames), we would represent an object `book` with two frames: the original frame (e.g.,`book_frame_raw`) and a second one computed by the physics engine (e.g.,`book_frame_corrected`). Such an approach leads to the robot's 3D model being cluttered with multiple frames and does not scale well.

Another example pertains to geometric task planning: a geometric task planner typically needs to reason over hypothetical future states of the environment ("What happens if I move this glass onto that pile of books?"). The planner generates many possible future states, which in turn might require further processing (for instance, running a physics simulation). Such a tool would benefit a flexible representation system, where models are derived from each other, with partial modifications and different timescales.

A third example relates to human-robot interaction scenarios where *perspective taking* is important (a prototypical example being the game 'I spy with my little eye', as implemented in [16]). Perspective taking is a cognitive skill that relies on the ability for an agent to take someone else's point of view to estimate what they see from their perspectives. Perspective taking has previously been implemented in robotics by temporarily placing virtual cameras at eye locations for each of the humans tracked by the robot [17]. While acceptable for simple cases, such an approach does not maintain truly independent spatio-temporal models of the environment for each agent, and in particular, it does not permit the representation of proper false-belief situations. On the contrary, separate, independent world models as implemented by UNDERWORLDS effectively support such a skill, which is an important precursor to research and implement human's mind modelling (i.e., a theory of mind) [18].

Lastly, geometric *pre-supposition accommodation* makes another interesting case for alternative worlds representation. *Pre-supposition accommodation* originally comes from linguistics, where it describes the mechanism by which *context is adjusted [...] to accept [...] a sentence that imposes certain requirements on the context in which it is processed* [19]. In the context of spatio-temporal representations, we call pre-supposition accommodation the ability of an agent to adjust its model so that it matches some contextual constraint. For instance, if A tells B to "catch the red balloon behind you", B might create a representation of an imaginary red balloon, placed behind her, even without actually observing the balloon: B *accommodates* the *pre-supposition* of a red balloon being present behind herself. Endowing robots with this capability has been touched upon by Mavridis et al. within their multi-modal *Grounded Situation Model* [2]. However, to the best of our knowledge, a general framework which would enable robots to accommodate spatial and temporal pre-suppositions by deriving imaginary worlds from existing ones has not been proposed so far.

UNDERWORLDS addresses this need and the main contribution of this work is a generic approach to **represent and share multiple parallel representations of the world**. UNDERWORLDS does so by allowing clients to clone existing worlds, modify them, and re-share them, without the cost of duplicating geometric data (as explained in section II). By organising clients in a network (Figure 1), worlds can be made dependent on each other, resulting in a loosely-coupled modular approach to spatio-temporal world representation that we call *cascading situation assessment*.

II. DESIGN AND ARCHITECTURE

A. Software architecture

Figure 1 depicts a typical UNDERWORLDS topology: a graph (that happens to be an *acyclic* graph on Figure 1, but does not have to be in the general case) of worlds, with clients connecting the worlds to each others.

1) Clients: Software components implementing accessing UNDERWORLDS worlds are called clients. Clients can both read and write onto the worlds they are connected to, and automatically see updates broadcast by other clients connected to the same world. To ensure data consistency, worlds can have many simultaneous readers, but only one writer at a given time.

UNDERWORLDS provides several standard clients (like a 3D visualisation tool or a physics engine simulator). Clients are however typically written by the end users, depending on the needs of one's specific architecture.

2) Worlds: Worlds are effectively distributed data structures composed of a scene graph representing the 3D ge-

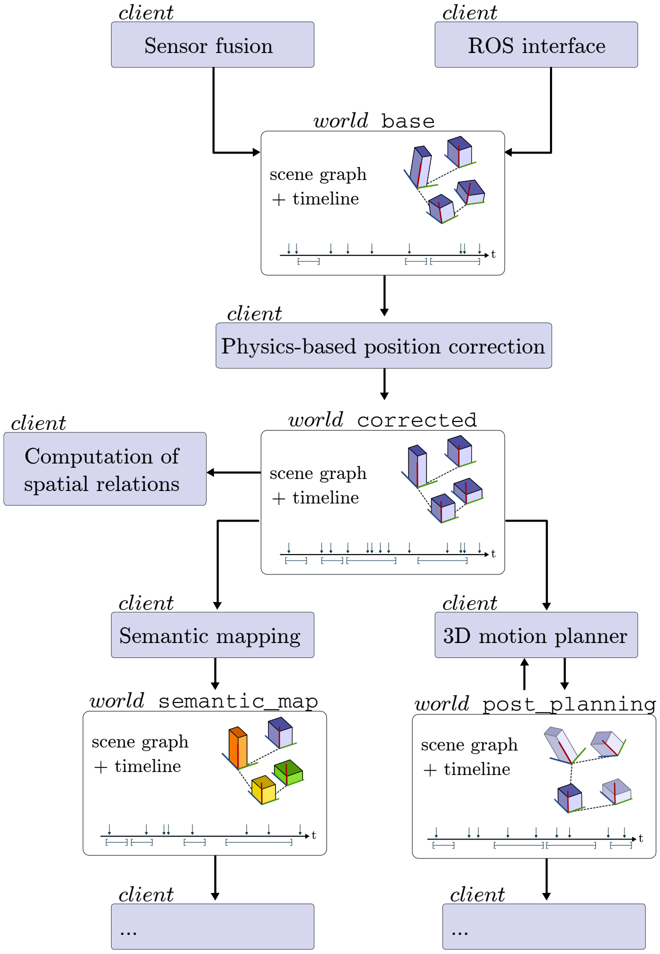

Fig. 1. Schema of a possible UNDERWORLDS network: eight *clients* (user-written & architecture specific; in blue) are sharing environment models through four independent *worlds* (made from joint spatial and temporal models). This architecture enables successive and modular refinement of the models (*cascading* situation assessment), effectively adapted to each client's needs.

ometry of the environment, and a timeline storing temporal events.

While each world is technically independent from all the others, dependencies (and therefore, coupling) arise between worlds from the clients' connections. For instance, filters effectively create a dependency between worlds. On Figure 1, the *Physics-based position correction* client creates a dependency between the world `base` (which represents here the result of raw sensor fusion) and the world `corrected` which would be a physically-consistent copy of `base`. As a result, an UNDERWORLDS network can also be seen as a dependency graph between worlds (where cyclic dependencies are permissible).

This architecture enables what we call *cascading situation assessment*: independent software components (the clients) build, refine and share successive models of the environment by a combination of filtering/transformations steps and model branching. A change performed by one client (for instance, a face tracker updates the pose of the human head) may thereby *cascade* to each of the downstream, dependent worlds.

3) Scenes: Worlds contain both a geometric model and a temporal model. The geometric model is represented as a scene graph. The scene graph has a unique root node, to which a tree of other nodes is parented.

Nodes in an UNDERWORLDS scene graph have three possible types: **objects** that represent concrete physical objects (typically with one or several associated 3D meshes); **entities** that represent abstract entities like reference frames or groups of objects; **perspectives** that represent viewpoints of the scene (like cameras or human gaze).

Every node has a unique ID, a parent, a 3D transformation relative to the parent and an optional name. *Object* nodes optionally store as well pointers to their associated meshes. Importantly, mesh data (or other geometric datasets like point clouds) are *not* stored within the nodes themselves. UNDERWORLDS represents geometric data as immutable data, identified by their hash value (preventing *de facto* data duplication). Nodes only store the hash corresponding to the desired geometric data, and the actual data is pulled from the server by the clients whenever they actually need it (for rendering for instance).

4) Timelines: Complementing the spatial representation encapsulated in the scene graph, each world also stores the world's *timeline*. This data structure is shared and synchronised amongst the clients in the same way as the scene graph. Clients can record and query both *events* (durationless states) and *situations* in the timeline, i.e., states with a start time and a (possibly open-ended) end time.

B. Distributed spatio-temporal models

UNDERWORLDS is not a monolithic piece of software. Instead, it stands for both a *network of interconnected clients* which manipulate spatial and temporal models of the robot environment (for instance, a motion planner, a object detection module, a human skeleton tracker, etc.), and for a client library that makes it possible to interface existing software components with the network.

Critically, the network is essentially hidden to the client: from the user perspective, the environment model is manipulated as a local data structure (see Listing 1). Modifications to the model are asynchronously synchronised with a central server (the `underworlded` daemon) and broadcast to every other client connected to the same world.

As previously mentioned, worlds are composite data structures comprised of a scene graph and a timeline. These data structures are synchronised using Google's gRPC message passing framework[2], ensuring high throughput, reliability and cross-platform/cross-language support. The UNDERWORLDS API is specifically discussed hereafter, in section III-A.

UNDERWORLDS is meant to broadcast complex environment representations (typically including large geometric datasets, like meshes) in real-time. UNDERWORLDS itself does not perform many CPU intensive tasks (CPU intensive

---

[2] `http://www.grpc.io/`

processing tasks – sensor fusion, physics simulation, etc.– are performed by the clients themselves) and as such, the performance bottleneck is essentially the network's data throughput. In that regard, one of the simple yet critical optimisations performed by UNDERWORLDS is automatic caching of mesh data. Mesh data are not transmitted when nodes are updated; only a hash value of the mesh data. The client can then request the full data whenever it is actually needed.

C. Time and space complexity analysis

UNDERWORLDS is fundamentally about distributing two datastructures: a scene graph (with nodes representing spatial entities) and a timeline (where events are stored as a flat list). Typical time and space complexities arise from these datastructures. In typical usage scenarios (where the number of nodes or events remain under a few hundred relatively small), the computational load to manipulate these datastructures is however dominated by the actual processings performed by the clients with the data. In the current implementation, scene graphs and timelines are stored in-memory. Were they required, serialization and persistent storage are not anticipated to be difficult to implement.

More interesting is the time complexity of distributing changes across an UNDERWORLDS network. With n the number of worlds and m the number of clients in an UNDERWORLDS network, the worst-case (when every world is a parameter of every client) time complexity of creating or updating a node and propagating the change across the network is $O(n \times m)$ (this effectively corresponds to the UNDERWORLDS server performing $n \times m$ requests to notify clients of the update). The space complexity is the same (as clients own a full copy of the worlds they monitor), except for mesh data whose space and time complexities are $O(1)$ (only the server stores the mesh data).

In the common case of one client performing a full update of a single world (with p nodes) at each time step, the complexity of propagating these changes across the network would be $O(p \times m)$. Figure 2 shows measured propagation time for one change across up to 20 cascading worlds.

III. API & CLIENTS

A. API

As mentioned, UNDERWORLDS uses Google's gRPC as message passing protocol. The protocol is explicitly defined (using the *protocol buffers*[3] interface definition language), and bindings to various languages and platforms can be automatically generated from the protocol definition file (as of Jan 2018, gRPC can generate bindings for C, C++, C#, Node.js, PHP, Ruby, Python, Go and Java, on Windows, Mac, Linux and Android). The cross-platform/cross-language support of gRPC is especially welcome in the academic context, as it offers ease and flexibility to plug a variety of pre-existing components into an UNDERWORLDS network.

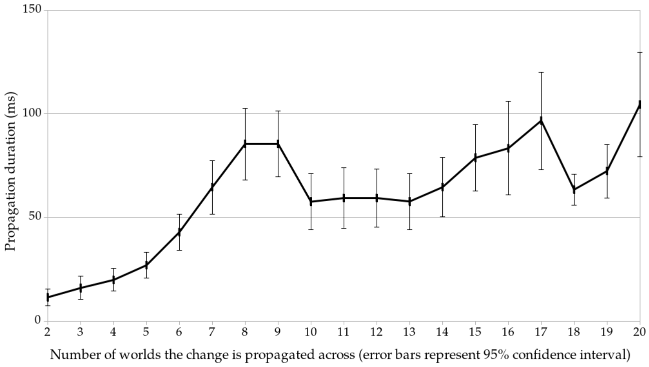

Fig. 2. Propagation times of one change (node creation) across n worlds. The test is performed by running $n-1$ pass-through filters that monitor one world and replicate any changes into the next world. Durations measured over 20 runs, performed on a 8 core machine.

However, the gRPC message passing layer is low-level with respect to the typical use of UNDERWORLDS (manipulation of asynchronous, distributed spatio-temporal models of the robot environment). In particular, the asynchronous fetching (and conversely, remote updating) of nodes and time-related objects is typically hidden from the user, and managed instead by the UNDERWORLDS client library.

UNDERWORLDS currently offers such a high-level client library for Python only (a C++ library is under development). Listing 1 gives a complete example of an UNDERWORLDS client performing simple filtering: the client continuously listens for changes in an input world, removes some objects (in this case, items whose volume is below a threshold), and forwards all other changes to an output world, effectively making the output world a copy of the input world with all smaller objects removed.

```python
import underworlds

# by default, connect to the server on localhost
with underworlds.Context("small_object_filter") as ctx:

    in_world = ctx.worlds["world1"]
    out_world = ctx.worlds["world2"]

    while True:

        in_world.scene.waitforchanges()

        for node in in_world.scene.nodes:
            if node.volume > THRESHOLD:
                out_world.scene.nodes.update(node)
```

Listing 1: Example of a simple yet complete UNDERWORLDS filter, written in Python: the client connects to the UNDERWORLDS network, blocks until the world `world1` changes, and only propagate nodes that match the condition to the world `world2`.

B. Standard Clients

The UNDERWORLDS package provides several standard clients to perform common tasks on UNDERWORLDS networks.

---

[3] `https://developers.google.com/protocol-buffers/`

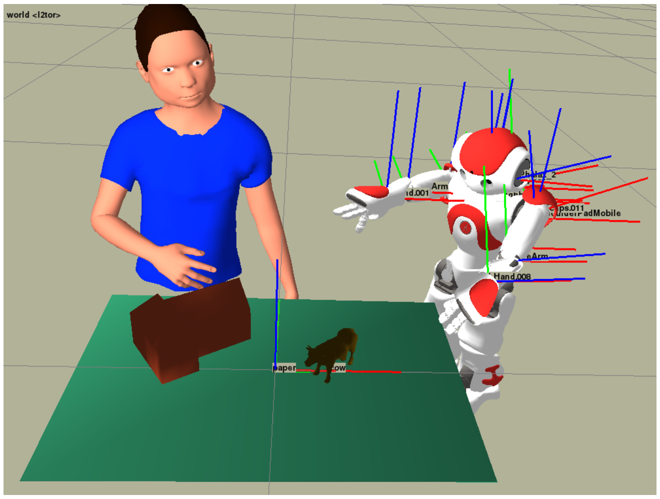

Fig. 3. Screenshot of the `uwds view` 3D visualisation and manipulation client. In this particular example, the 3D meshes have been pre-loaded using `uwds load`. Their positions are then updated at run-time using the robot's sensors and proprioception (joint state).

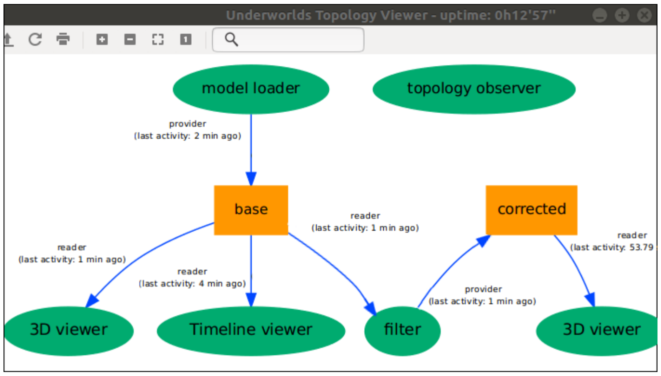
Underworlds Topology Viewer - uptime: 0h12'57"
model loader
topology observer
provider
(last activity: 2 min ago)
base
corrected
reader
(last activity: 1 min ago)
reader
(last activity: 53.79
reader
(last activity: 1 min ago)
reader
(last activity: 4 min ago)
provider
(last activity: 1 min ago)
3D viewer
Timeline viewer
filter
3D viewer

Fig. 4. Screenshot of the network topology introspection tool, with arbitrary examples of worlds (represented as boxes) and clients (ellipses). CLients are connected to the worlds either as *readers* or *providers* of data. UNDERWORLDS introspection features make it possible to also visualise when clients were last active.

1) 3D Visualisation and manipulation: Interestingly, while UNDERWORLDS deals with 3D geometries and scenes, it does represent 3D entities purely as data structures; no visual representation is involved (and as such, the UNDERWORLDS server and core libraries do not depend on any graphics library like OpenGL). However, for all practical purposes, the ability to visualise the content of a scene is desirable. UNDERWORLDS provides a standard client, `uwds view`, that performs real-time 3D rendering of worlds, using OpenGL (Figure 3).

This tool also supports basic object manipulations (translations, rotations), that are broadcast to the other UNDERWORLDS clients connected to the same world.

Assets loading: Often, objects manipulated by the robot have known meshes with corresponding CAD models that can be conveniently pre-loaded. In these cases, UNDERWORLDS provides a tool, `uwds load`, that loads a mesh into a UNDERWORLDS network (and optionally, creates a node) from a large range of 3D formats (including Collada, FBX, OBJ, Blender)[4].

2) Physics simulation: When perception modules provide objects localisation, the physical consistency of the locations is not typically enforced. For instance, objects that are supposed to lay on a table might be slightly above (or inset into) the table; or when dropping an item into a box, the robot can not update the location of the item anymore as it becomes occluded.

These issues can be alleviated by relying on a physics simulation to stabilise the position of objects: natural physics (including gravity) are simulated for a short amount of time (up to one second) ahead of time, and the objects' positions are updated accordingly. To this end, UNDERWORLDS provide a standard filter, the `physics_filter`, based on the Bullet RT physics simulation and the `pybullet`[5] library. It generates an output world that mirrors its input world after a specific duration of physics simulation, the physical properties of objects (including mass, friction, inertia) being provided from standard URDF descriptions.

3) Introspection and debugging: UNDERWORLDS provides a range of tools to inspect a running network. Graphical tools (`uwds explorer` and `uwds timeline`, see Figure 4) provide a user-friendly overview of the system's graph with the connections between the clients and the worlds, as well as their activity.

Specialised command-line tools are also available to list the worlds and their content (`uwds ls`) at run-time, or to display detailed information for a specific node (`uwds show`).

4) Interface with ROS: UNDERWORLDS is meant to integrate as easily as possible into existing robot architectures, and interfaces transparently with ROS' TF frame system through the `uwds tf` client.

The `uwds tf` client continuously monitors the ROS TF tree, and mirrors TF frames as nodes in the desired UNDERWORLDS world. A node is first created if none matches a given TF frame, and its transformation is subsequently updated, mirroring the TF frame. A regular expression can be provided to only mirror a subset of the TF tree into UNDERWORLDS.

Currently, the process is unidirectional: the `uwds tf` client performs TF to UNDERWORLDS updates, but not the reverse.

C. Spatial Reasoning and Perspective Taking

Spatial reasoning [20] is a field in its own right, and has been used for natural language processing for applications such as direction recognition [21], [22] or language grounding [23]. Other examples in human-robot interaction include

---

[4] The underlying import capability is provided by the ASSIMP library. `http://assimp.sourceforge.net/`

[5] `https://pybullet.org/`

Ros et al. [17], [16] which has recently been integrated into a full architecture for autonomous human-robot interaction [10].

UNDERWORLDS provides an exemplary client (`spatial_relations`) to compute both allo-centric (independent of the viewpoint like `isIn` or `isOn`) and ego-centric (i.e., viewer-dependent, like `inFrontOf` or `leftOf`) spatial relations between objects. Other libraries, like QSRLib [24], that implement computational models of Qualitative Spatial Relations, could be trivially combined with UNDERWORLDS to provide more advanced geometric analysis. Future developments will also include the results of the more basic research on spatio-temporal reasoning for robotics, led by de Leng and Heintz [25].

UNDERWORLDS also implements an efficient algorithm to assess object visibility from a specific viewpoint (i.e., from a given *perspective* node). The algorithm (color picking) enables fast (single pass) computation of the visibility of every object in the scene, while providing control regarding how many pixels should be actually visible for the object to be considered globally visible. The command-line tool `uwds visibility` returns the list of visible objects from the point of view of each camera in a given world, and UNDERWORLDS also provides the helper class `VisibilityMonitor` to programmatically access visibility information.

When integrated into a filter node, visibility computation allows easy creation of new worlds representing the estimated perspectives of the different agents.

IV. APPLICATION EXAMPLE: PERSPECTIVE-AWARE JOINT ACTIONS

UNDERWORLDS is being used within the large European project MuMMER[6] for service robots to compute visibility and knowledge about objects, places and agents within a mall environment.

We present here a simplified scenario, yet representative of situations which are processed in real-time by MuMMER robots: two humans and a robot are looking at a table and have to coordinate joint actions (pick and place). One object on the table (the green box in Figure 5) is only visible to one human and the robot, but hidden to the second human. The robot needs to take into account this fact to generate appropriate and legible joint manipulation actions. Figure 5 illustrates the topology of the UNDERWORLDS network that we use to this end.

A first client, *static_env_provider*, provides the environment models and allows to build a first `ENV` world where static objects, furnitures and walls are present. Then, three worlds cascade through three (independent) clients: *robots_state_monitor* augments `ENV` with the robot state (using underneath the ROS robot state publisher node) and broadcast a new world `ENV_ROBOTS`. *objects_monitor* then recognises and adds the dynamic objects (using `ar_track_alvar`[7]). *humans_monitor* finally detects and continuously updates the humans poses (using [26]). It broadcasts a world called `BASE` that contains as a result the static environment, the robots, the dynamic objects and the detected humans.

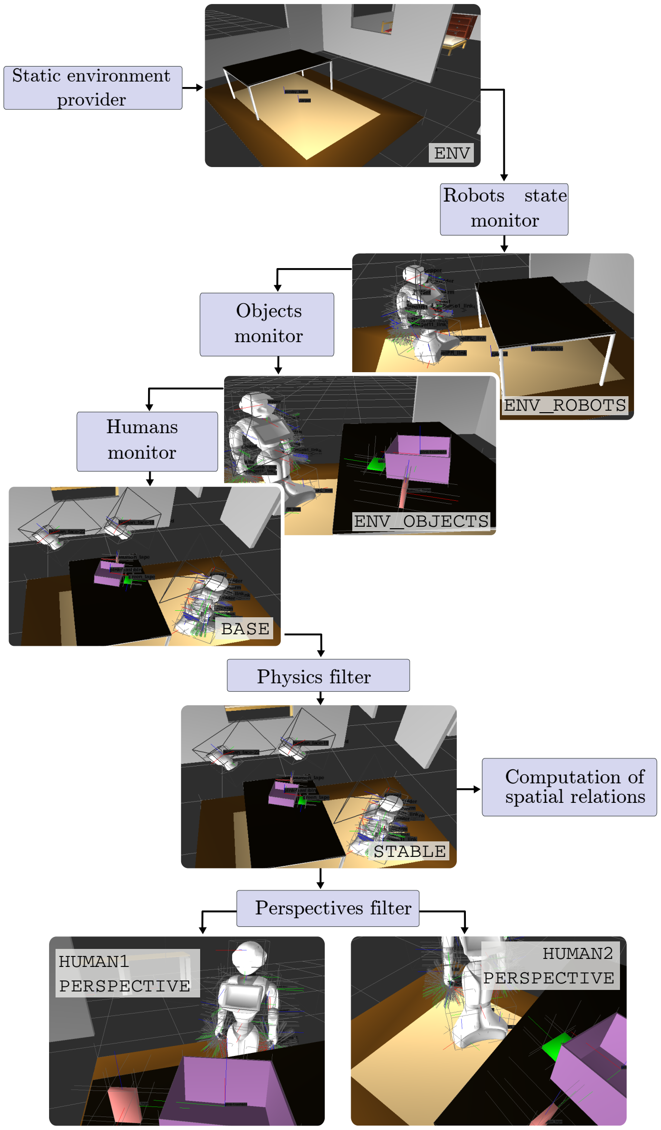

Fig. 5. Schema of the UNDERWORLDS architecture used in the MuMMER project. Clients read and generate the worlds `ENV` → `ENV_ROBOT` → … → `HUMAN*_PERSPECTIVE`. The last two worlds `HUMAN{1,2}_PERSPECTIVE` represent the immediate visual perspective of each of the humans, as well as their past visual perceptions. As such, they are the *visual memories* of the humans, that the robot can rely on when making decisions.

The world `BASE` goes through a *physics filter* client (as explained in section III-B.2) to obtain the `STABLE` world where all elements are present with physically-consistent locations. This physically-correct world is used by the *computation_of_spatial_relations* client to compute spatial relations such as `onTop`, `isIn` or `isAbove` (see Section III-C).

The world `STABLE` is also used by a *perspectives_filter* client to compute the different visual perspectives of each agent (in our case: human 1, human 2 and the robot itself).

---

[6] http://www.mummer-project.eu

[7] http://wiki.ros.org/ar_track_alvar

In addition to a 3D rendering of the input world from the perspective of the agent, it aggregates the history of what was visible to the agent at a given point in time. As such, it does not only offer a snapshot of the agent visual perspective at the current time but also acts as the visual memory of each agent.

With this network, the robot can easily compute that an object on the table is only seen by the human 1 and not the human 2; additionally, if human 1 moves in a position where the object is not visible anymore to him, the *perspective_filter* will maintain the knowledge that the human had seen it (and keep the last position where it has been seen).

UNDERWORLDS makes it possible to implement such a geometric reasoning pipeline in a fully decoupled way, and each intermediary world can be easily introspected at run-time. This example shows how UNDERWORLDS facilitates the implementation and debugging of complex spatio-temporal reasoning pipelines.

We are currently deploying a similar network in the framework of the European project MuMMER where a Pepper robot handles interactive situations in a large shopping centre in Finland. One of the situation is a guiding task where Pepper help people to find their route by pointing them landmarks and explaining them how to reach a destination. To be effective, this helping behaviour needs to be aware of the visual perspective of the human. UNDERWORLDS facilitates the implementation of such a spatio-temporal reasoning pipeline, where perception and high-level reasoning (including complex, human-aware reasoning) have to be tightly integrated. Because of the decoupling of each of the clients in the network, UNDERWORLDS also practically supports software development spread across multiple partners in different countries, with different expertise.

V. DISCUSSION AND CONCLUSION

A. Relation to existing robotic middleware

Like traditional robotic middleware, UNDERWORLDS offers a form of distributed computation based on message passing. However, it distinguishes itself from existing middlewares (including ROS extensions like DyKnow [27]) in significant ways. Most importantly, UNDERWORLDS purposefully does not offer any *general* capability to distribute computation and data streams amongst independent components: it focusses specifically on distributing environment models, both spatial (geometric models) and temporal (events and situations). In that sense, UNDERWORLDS really is a *distributed datastructure* that addresses the specific needs of spatio-temporal modelling, including the modelling of hypothetical, alternative world models, something that traditional middlewares like ROS do not address adequately. Second, and as presented above, UNDERWORLDS offers specific mechanisms for the representation and manipulation of alternative world models that are not directly achievable with traditional tools.

While using standard middleware as *underlying transport* for UNDERWORLDS would be technically feasible and relatively easy to implement, it does not offer any clear advantage over lighter and dedicated message passing libraries like ZeroMQ or gRPC (the later being the one used by UNDERWORLDS).

B. Future work

As illustrated in section IV, UNDERWORLDS is already deployed and used on the field. Several features are however still under development.

1) Representation capabilities: as presented in section II, the current version of UNDERWORLDS allows to represent *objects*, *abstract entities* like groups and *perspectives*. *Fields* are also part of the UNDERWORLDS design, but are not yet implemented. Fields are commonly used to represent continuously-valued spatial entities. Fields might or might not be spatially bounded. Examples include the working space of a robot arm (spatially bounded), the field of view of a camera (spatially bounded), proxemics (potentially unbounded). We plan to represent fields in UNDERWORLDS using the memory-efficient octomaps [28] or NDT-OM maps [29]. Similarly to geometric data,these datastructures will not be directly stored with the nodes (nodes will refer to them through handles), but unlike geometric data, they will not be treated as immutable datasets by the server, permitting real-time updates.

Representation of uncertainty: currently node positions are stored as 4×4 transformation matrices, relative to the node parent. This representation is efficient, and conveniently matches traditional representation systems (including ROS TF frames or OpenGL transformations). However, the explicit management of uncertainties is instrumental to many robotic applications, and we plan to add full support for position uncertainties to UNDERWORLDS. We plan to add this support by adding a pose covariance matrix to the nodes, and equipping the different UNDERWORLDS helper tools with corresponding support (like covariance ellipses visualisation in `uwds view`).

2) Implementation and Integration: we plan to continue to improve the integration of UNDERWORLDS into existing software architectures. A short-term goal is to provide excellent C++ support, with a high-level, user-friendly C++ client library. This is critical for a broader adoption of UNDERWORLDS within the robot community. Support for other languages might follow, depending on demands and open-source contributions.

C. Conclusion

We have introduced UNDERWORLDS, a novel framework for shared and composable spatio-temporal representations of a robot's world. The key contributions of our approach are: a composite data structure for environment representation within a robotic software architecture, made of a scene graph and a timeline; a mechanism to efficiently and transparently share this data structure amongst a set of clients (the software modules of the robot); a cascading architecture permitting the explicit of representation of alternative states of the world while maintaining a network of dependencies.

We have additionally presented a concrete instantiation of a system relying on UNDERWORLDS for its representation needs, and we have sketched future directions of development.

We believe this work can practically support existing robotic architectures with state-of-the-art spatio-temporal representation capabilities. We also hope that this line of research can lead to a better understanding of the representation needs of modern robotic systems, and participate to the emergence of a possible common representation platform for robots, building on previous formalisation efforts like the RSG-DSL domain specific language [30].

ACKNOWLEDGMENT

This work has been supported by the EU H2020 Marie Skłodowska-Curie Actions project DoRoThy (grant 657227), the EU H2020 MuMMER project (grant 688147) and the EU H2020 L2TOR project (grant 688014).

REFERENCES

[1] H. Jacobsson, N. Hawes, G.-J. Kruijff, and J. Wyatt, "Crossmodal content binding in information-processing architectures," in *Proceedings of the 3rd ACM/IEEE International Conference on Human Robot Interaction*. New York, NY, USA: ACM, 2008, pp. 81–88.

[2] N. Mavridis and D. Roy, "Grounded situation models for robots: Where words and percepts meet," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006.

[3] A. Nüchter and J. Hertzberg, "Towards semantic maps for mobile robots," *Robotics and Autonomous Systems*, vol. 56, no. 11, pp. 915 – 926, 2008.

[4] C. Galindo, J. Fernández-Madrigal, J. González, and A. Saffiotti, "Robot task planning using semantic maps," *Robotics and Autonomous Systems*, vol. 56, no. 11, pp. 955–966, 2008.

[5] N. Blodow, L. C. Goron, Z.-C. Marton, D. Pangercic, T. Rühr, M. Tenorth, and M. Beetz, "Autonomous semantic mapping for robots performing everyday manipulation tasks in kitchen environments," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, San Francisco, CA, USA, September, 25–30 2011.

[6] C. Lörken and J. Hertzberg, "Grounding planning operators by affordances," in *International Conference on Cognitive Systems (CogSys)*, 2008, pp. 79–84.

[7] K. Varadarajan and M. Vincze, "Ontological knowledge management framework for grasping and manipulation," in *IROS Workshop: Knowledge Representation for Autonomous Robots*, 2011.

[8] E. A. Sisbot, R. Ros, and R. Alami, "Situation assessment for human-robot interactive object manipulation," in *2011 RO-MAN*, July 2011, pp. 15–20.

[9] G. Milliez, M. Warnier, A. Clodic, and R. Alami, "A framework for endowing an interactive robot with reasoning capabilities about perspective-taking and belief management," in *The 23rd IEEE International Symposium on Robot and Human Interactive Communication*, 2014, pp. 1103–1109.

[10] S. Lemaignan, M. Warnier, E. A. Sisbot, A. Clodic, and R. Alami, "Artificial cognition for social human-robot interaction: An implementation," *Artificial Intelligence*, 2016.

[11] M. Beetz, M. Tenorth, and J. Winkler, "Open-EASE – a knowledge processing service for robots and robotics/ai researchers," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 1983–1990.

[12] M. Naef, E. Lamboray, O. Staadt, and M. Gross, "The blue-c distributed scene graph," in *Proceedings of the workshop on Virtual environments 2003*. ACM, 2003, pp. 125–133.

[13] S. Blumenthal, H. Bruyninckx, W. Nowak, and E. Prassler, "A scene graph based shared 3d world model for robotic applications," in *2013 IEEE International Conference on Robotics and Automation*, May 2013, pp. 453–460.

[14] P. Bustos, L. J. Manso, J. P. Bandera, A. Romero-Garcés, L. V. Calderita, R. Marfil, and A. Bandera, "A unified internal representation of the outer world for social robotics," in *Robot 2015: Second Iberian Robotics Conference*. Springer, 2016, pp. 733–744.

[15] M. Beetz, D. Jain, L. Mösenlechner, and M. Tenorth, "Towards performing everyday manipulation activities," *Robotics and Autonomous Systems*, vol. 58, no. 9, pp. 1085–1095, 2010.

[16] R. Ros, S. Lemaignan, E. A. Sisbot, R. Alami, J. Steinwender, K. Hamann, and F. Warneken, "Which one? grounding the referent based on efficient human-robot interaction," in *19th IEEE International Symposium in Robot and Human Interactive Communication*, 2010.

[17] R. Ros, E. A. Sisbot, R. Alami, J. Steinwender, K. Hamann, and F. Warneken, "Solving ambiguities with perspective taking," in *Proceedings of the 5th ACM/IEEE international conference on Human-robot interaction*. IEEE Press, 2010, pp. 181–182.

[18] S. Lemaignan and P. Dillenbourg, "Mutual modelling in robotics: Inspirations for the next steps," in *Proceedings of the 2015 ACM/IEEE Human-Robot Interaction Conference*, 2015.

[19] K. Von Fintel, "What is presupposition accommodation, again?" *Philosophical perspectives*, vol. 22, no. 1, pp. 137–170, 2008.

[20] J. O'Keefe, *The Spatial Prepositions*. MIT Press, 1999.

[21] T. Kollar, S. Tellex, D. Roy, and N. Roy, "Toward understanding natural language directions," in *HRI*, 2010, pp. 259–266.

[22] C. Matuszek, D. Fox, and K. Koscher, "Following directions using statistical machine translation," in *Proceedings of the International Conference on Human-Robot Interaction*. ACM Press, 2010.

[23] S. Tellex, "Natural language and spatial reasoning," Ph.D. dissertation, MIT, 2010.

[24] Y. Gatsoulis, M. Alomari, C. Burbridge, C. Dondrup, P. Duckworth, P. Lightbody, M. Hanheide, N. Hawes, D. Hogg, A. Cohn *et al.*, "Qsrlib: a software library for online acquisition of qualitative spatial relations from video," Tech. Rep., 2016.

[25] D. de Leng and F. Heintz, "Qualitative spatio-temporal stream reasoning with unobservable intertemporal spatial relations using landmarks." in *AAAI*, 2016, pp. 957–963.

[26] V. Khalidov and J.-M. Odobez, "Real-time multiple head tracking using texture and colour cues," Idiap, Idiap-RR Idiap-RR-02-2017, 2 2017.

[27] D. de Leng and F. Heintz, "DyKnow: A Dynamically Reconfigurable Stream Reasoning Framework as an Extension to the Robot Operating System," in *IEEE Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, 2016, pp. 55–60. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-132266

[28] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: An efficient probabilistic 3d mapping framework based on octrees," *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, 2013.

[29] J. P. Saarinen, H. Andreasson, T. Stoyanov, and A. J. Lilienthal, "3d normal distributions transform occupancy maps: An efficient representation for mapping in dynamic environments," *The International Journal of Robotics Research*, vol. 32, no. 14, pp. 1627–1644, 2013.

[30] S. Blumenthal and H. Bruyninckx, "Towards a domain specific language for a scene graph based robotic world model," *arXiv preprint arXiv:1408.0200*, 2014.