frontiers | Frontiers in Virtual Reality

TYPE Original Research
PUBLISHED 03 December 2024
DOI 10.3389/frvir.2024.1471579

OPEN ACCESS

EDITED BY
Anna Kobusinska,
Poznań University of Technology, Poland

REVIEWED BY
Ruoyu Zhao,
Nanjing University of Aeronautics and Astronautics, China
M. Shahid Anwar,
Gachon University, Republic of Korea
Jay Lofstead,
Sandia National Laboratories (DOE), United States

*CORRESPONDENCE
Sai Anirudh Karre,
saianirudh.karri@research.iiit.ac.in

RECEIVED 27 July 2024
ACCEPTED 31 October 2024
PUBLISHED 03 December 2024

CITATION
Karre SA and Reddy YR (2024) Model-based approach for specifying requirements of virtual reality software products.
Front. Virtual Real. 5:1471579.
doi: 10.3389/frvir.2024.1471579

Model-based approach for specifying requirements of virtual reality software products

Sai Anirudh Karre* and Y. Raghu Reddy

Software Engineering Research Center, International Institute of Information Technology Hyderabad, Hyderabad, India

Introduction: Gathering requirements for developing virtual reality (VR) software products is a labor-intensive process. It requires detailed elicitation of scene flow, articles in the scene, action responses, custom behaviors, and timeline of events. The slightest change in requirements will escalate the design and development costs. While most VR practitioners depend on conventional software engineering (SE) requirement-gathering techniques, there is a need for novel methods to streamline VR software development. With severe software platform fragmentation and hardware volatility, VR practitioners need assistance specifying non-volatile requirements for a minimum viable VR software product.

Methods: To address this gap, we present virtual reality requirement specification tool (VReqST), a model-based requirement specification tool for developing virtual reality software products.

Results: Using VReqST, requirement analysts can specify the requirements for both simple and complex multi-scene VR software products and virtual environments (VEs).

Discussion: VReqST is customizable and competent in illustrating custom requirements for new locomotion, colocation, teleportation algorithms, etc. We worked with the VR community from the industry for adoption and feedback. We revised and included the desired features based on inputs from the VR community and gathered their observations on the overall impact of VReqST in practice.

KEYWORDS

virtual reality, requirement specification, meta model, model-driven development, software modeling, metaverse

1 Introduction

Virtual reality (VR) technology can potentially revolutionize the future of work. Several big companies, including Meta, NVIDIA, Microsoft, Apple, and SAP, have already created proofs-of-concept using immersive solutions for remote collaborative work and complex task completion (Gramlich, 2022). These VR enterprise solutions have become essential in various industries such as healthcare, manufacturing, training, IT, energy, and retail. They provide application-specific domain practitioners with immersive experiences that help them comprehend complex activities through simulations. In the post-pandemic era, immersive solutions are increasingly embraced to enhance productivity in enterprise industries by fostering innovation. This trend has also given rise to the "***metaverse***," three-dimensional virtual experience software that allows people to socialize, learn,

FIGURE 1
VR bowling alley game in a developer mode (UNITY Engine)—point-of-view 1 and 2.

collaborate, implement, work, play, and experience emotions virtually using real-world metaphors. Despite the numerous advantages, the overall VR product development process for developing enterprise VR software is plagued by significant technological and practitioner challenges. Recent studies by Anwar et al. (2024) have also illustrated the quality of experience in adapting, augmenting, and producing standards for immersive environments. VR software product development has been observed to employ different sets of practices compared to traditional software product development (Karre et al., 2019), resulting in the following challenges.

- Platform fragmentation: Almost all the VR software development toolkits (SDK) do not prioritize portability and cross-platform compatibility with respective head-mounted devices (HMDs). This forces the VR community to re-build their VR code for various HMDs (Group, 2019).
- Volatile hardware: The computational capabilities of existing HMDs deteriorate relative to newer models due to platform fragmentation. Thus necessitating frequent hardware upgrades to leverage the features of the latest SDK features. To stay relevant in the market, the VR community has to either commit to a particular HMD-SDK version or continually rebuild their VR products.
- Poor tool support: VR developer tools suffer from inadequate tool support. The lack of standardized naming conventions in these tools hinders a common understanding of the VR domain. Consequently, most VR tools heavily rely on their own unique VR model information. To mitigate confusion, the VR community has gravitated toward using a select few tools. However, the integration of VR products poses a significant risk to the community due to portability issues (Nebeling and Speicher, 2018).
- Multi-modality: VR is a multi-modal technology (Martin et al., 2022) that necessitates thorough requirement gathering in order to create realistic virtual environments. Various modalities such as gestures, gaze, auditory stimuli through haptic devices, kinesthetic feedback (related to proprioception), temperature, pressure, and pain-sensing are employed in VR to cater to different use cases. To ensure the realism of VR software products, it is crucial for requirement analysts to conduct comprehensive requirement gathering.
- Risky assumptions: The development cost of a VR product can significantly increase due to obscure and abstract prototype designs. VR developers cannot make assumptions about the VR scene with complete certainty. Unclear and hypothetical use-case stories, articles, action responses, specific behaviors, and timelines of events related to the VR scenes can result in misleading outcomes. Consequently, this leads to wastage in VR software development.
- Stakeholders: The VR developer community comprises various stakeholders such as virtual environment (VE) designers, VR developers, acoustic engineers, interaction animators, usability practitioners, VE testers, and maintenance engineers, among others, who may not be aware of the traditional enterprise software development methods. It is crucial to note that miscommunication among these stakeholders or straying from the specified requirements can result in substantial development setbacks and the need for additional work.

The aforementioned factors can be alleviated by obtaining clear and comprehensive specifications, thereby preventing the need for rework and reducing the cost associated with development. It has been noted that proficient requirement engineering methodologies can result in enhanced software quality (Filho and Kochan, 2001). Conversely, the process of eliciting requirements in virtual reality is perceived as arduous and necessitates meticulous documentation (Karre et al., 2023). Figure 1 illustrates an example of a VR bowling alley game (Kandhari, 2023) in a UNITY development editor mode to understand the underlying details required to elicit and specify requirements for VR software. We can observe articles like pins, balls, gutters, scoreboards, light sources, and walls populating in the three-dimensional scene. Each article holds distinct VR-specific properties like layout information, scale, material, gravity, mass, audio, ray-casting, object on collision, object rigidity, light sources, motion, rotation, occlusion, angular drag, object kinematic properties, interpolation of an object, wobble effect of object, VR participant locomotion, environment depth, scene rendering, synchronous–asynchronous VR participant task-action responses, control flow, and data flow of overall events. In comparison to two-dimensional or three-dimensional graphics, the complexity of requirements for VR software is significantly higher. Consequently, the requirement engineering process for VR scenes becomes multi-faceted and intricate. In order to facilitate the process

of requirement engineering for the development of VR products, we introduce a model-based tool called the virtual reality requirement specification tool (**VReqST**). This tool adopts a model based template approach, which means that it provides a structured framework for capturing and organizing requirements for the VR technology domain. It is designed to assist the VR community in specifying requirements for the creation of enterprise VR software.

2 Materials and methods

2.1 Virtual reality domain model template

Why is a model-based understanding of the VR technology domain required?—a virtual reality software system strives to induce targeted behavior in an organism (predominantly a human or an animal) using artificial sensory simulation (LaValle, 2020). VR hardware and software have evolved independently, causing disparity in the overall evolution of VR as a domain. Consequently, it has become difficult for VR practitioners to build portable and cross-platform VR products. Several attempts have been made to standardize VR as a domain through standards like Virtual Reality Modeling Language (VRML) by W3C (1997), (COLLAborative Design Activity (COLLADA) by Sony Computer Entertainment Inc. (2004), O3D by Google Inc. (2010), Filmbox (FBX), GL Transmission Format (glTF), and OpenXR. These standards have failed due to the lack of interoperability support between VR software and VR hardware (Brennesholtz, 2017). Following are few detailed insights into these prominent standards and the reasons for their adoption failure.

VRML (W3C, 1997) is a file format for describing three-dimensional (3D) interactive vector graphics, designed particularly for the web. It allows the creation of 3D scenes and objects to be viewed in a web browser or a standalone VRML browser, called a "player." VRML files can be created and edited using various 3D modeling software and can include textures, lighting, physics properties, animations, and behaviors. VRML has been replaced by X3D (ISO/IEC 19775-1) as the standard for 3D web graphics.

O3D was a web-based 3D graphics application programming interface (API) developed by Google as an alternative to Flash and other 3D technologies. It was designed to enable the creation and display of interactive 3D graphics in web browsers without the need for additional plugins or software. O3D provided a JavaScript API for creating and manipulating 3D scenes, objects, and animations and supported features such as lighting, texturing, and physics. O3D was intended to make it easier for web developers to create and share 3D content and to provide a more immersive and interactive web experience for users.

COLLADA is a royalty-free, XML-based file format for 3D digital assets. It was developed by the Khronos Group, an industry consortium focused on the creation of open standards for 3D graphics, and is used for exchanging 3D models and scenes between different applications and platforms. COLLADA supports a wide range of 3D modeling features, including geometry, textures, lighting, animations, and physics. It is designed to be platform-independent and can be used to exchange 3D assets between different operating systems, hardware platforms, and software applications. COLLADA files can be exported and imported using a variety of 3D modeling tools, game engines, and other software programs, making it a popular choice for interoperability in the 3D graphics industry. COLLADA is often used for creating and sharing 3D models and scenes for use in real-time 3D applications, such as games, simulations, and virtual reality experiences. It is also used in the film and television industry for creating 3D visual effects and animations.

Despite initial interest and support, all these standards failed to gain widespread adoption due to several challenges, as discussed below:

- Limited browser support: At the time of VRML's and O3D's introduction, few web browsers supported the format, making it difficult for users to view and interact with VRML content.
- Complexity: COLLADA is a complex and powerful file format that supports a wide range of 3D modeling features. This complexity can make it difficult for some users and developers to work with and can lead to issues with interoperability between different applications and platforms.
- Performance issues: VRML and O3D files could be large and complex, leading to slow loading times and poor performance on many systems.
- Lack of standardization: VRML lacked a clear and consistent set of standards, making it difficult for developers to create consistent and reliable content.
- Limited authoring tools: There were few user-friendly tools available for creating VRML content, making it difficult for non-technical users to create and share their own 3D models.
- Competition: VRML faces competition from other technologies such as Flash and Java 3D, while O3D competes with WebGL, Three.js, and Unity. Similarly, COLLADA faces competition from other 3D file formats, such as FBX and glTF, which offer similar capabilities and have gained wider adoption, further reducing the appeal of VRML, O3D, and COLLADA.
- Limited use cases: VRML and O3D were primarily used for creating 3D models and environments for the web and did not have a wide range of other applications, limiting its appeal.
- Lack of community and developer support: COLLADA and O3D did not have strong community or developer support, making it difficult for the technology to gain momentum and attract a critical mass of users and developers.

On other end, formats like FBX and glTF and have gained wide popularity due to their proprietary in nature.

FBX is a proprietary file format used for exchanging 3D digital assets between different applications and platforms. It was developed by Kaydara and owned by Autodesk Inc., which was used for exchanging 3D models, animations, and other assets between different 3D modeling tools, game engines, and other software programs. FBX supports a wide range of 3D modeling features, including geometry, textures, lighting, animations, and physics. It is designed to be platform-independent and can be used to exchange 3D assets between different operating systems, hardware platforms, and software applications. FBX files can be exported and imported using a variety of 3D modeling tools, game engines, and other software programs, making it a popular choice for

interoperability in the 3D graphics industry. It is often used for creating and sharing 3D models and scenes for use in real-time 3D applications, such as games, simulations, and virtual reality experiences. It is also used in the film and television industry for creating 3D visual effects and animations. As it is a proprietary format, it is not open to the public. Thus, users must have a license to use the FBX software development toolkit and underlying tools for working with this format. However, FBX is widely supported by many 3D modeling tools and game engines and is a popular choice for interoperability in the 3D graphics industry.

glTF, on the other hand, is an open, royalty-free file format for 3D digital assets. It was developed by the Khronos Group, an industry consortium focused on the creation of open standards for 3D graphics, and is used for exchanging 3D models and scenes between different applications and platforms. glTF is designed to be lightweight and efficient, with a focus on real-time 3D applications, such as games, simulations, and virtual reality experiences. It supports a wide range of 3D modeling features, including geometry, textures, lighting, animations, and physics, and is designed to be platform-independent and easy to use. glTF is an open standard, which means that it is freely available to the public and can be used and implemented by anyone. It is supported by a wide range of 3D modeling tools, game engines, and other software programs and is becoming an increasingly popular choice for interoperability in the 3D graphics industry. glTF is often compared to other 3D file formats, such as COLLADA and FBX, which offer similar capabilities. However, glTF has several advantages over these formats, including its lightweight and efficient design, its focus on real-time 3D applications, and its open and royalty-free status. As a result, glTF is gaining popularity as a modern and flexible file format for exchanging 3D assets between different applications and platforms.

In contrast, despite being proprietary (in the case of FBX) and open, royalty-free (in the case of glTF), they are not widely acceptable due to volatility in their underlying meta-model of the virtual reality technology domain. For example, consider a human with little or no awareness of the interference (LaValle, 2020). Such a human, through a software system, aspires to simulate a real-world environment as an immersive three-dimensional (3D) environment, primarily using 3D computer-generated graphics. Various human sensory aspects like auditory, haptic (through force), olfactory, vision, human motor, proprioception (body position and movement), and cognitive capacities are experienced through this system. *Immersion* and *presence* are ideal states of a VR software system but are not necessary prerequisites for engagement and interaction. Despite the increase in the computation power of VR hardware, VR software systems do not adequately achieve realism through underlying VR software. VR hardware and VR software have evolved independently, leading to severe platform fragmentation. Such challenges still continue with FBX and glTF standards. Recently, attempts have been made to support interoperability between VR software and hardware using OpenXR standard APIs (Group, 2019).

OpenXR is an open standard for virtual reality and augmented reality devices, developed by the Khronos Group, an industry consortium focused on the creation of open standards for 3D graphics. It provides a common, cross-platform API for accessing and controlling VR and AR devices, allowing developers to create applications that can run on a wide range of devices without the need for custom integration. It was initially designed to simplify the development and deployment of VR and AR applications by providing a single, unified API that can be used across different devices and platforms. This allows developers to create applications that can run on a wide range of devices without the need to write custom code for each device or platform. OpenXR is an open standard, which means that it is freely available to the public and can be used and implemented by anyone. It is supported by a wide range of VR and AR device manufacturers, software companies, and other organizations and is becoming an increasingly popular choice for cross-platform development in the VR and AR industry. OpenXR is still in the process of being developed and is not yet a final, stable standard. However, it is expected to be released in the near future and is already being used by some developers and organizations for cross-platform VR and AR development.

2.2 Attributes for VR software development

Overall, the available standards are either in draft or require more comprehensive usage/feedback within the VR practitioner community. Various commercial VR providers proposed and practiced variants of the VR conceptual model. These models are open-ended and do not permit interoperability to meet the necessities of a bare-minimum VR software system. At a minimum, a VR software system should depict components like scenes, scene objects, cameras, action responses, and behavior outcomes. However, the existing conceptual models do not provide common control and data flow to facilitate constructing a bare-minimum VR software system. Thus, we address the problem by formulating the following research questions:

> **RQ:** What constitutes a meta-model of a bare minimum VR software system?

Conceptualizing a meta-model for VR systems requires a thorough understanding of VR as a domain in addition to understanding VR systems in application domains like healthcare and banking. We made an unsuccessful attempt by conducting a systematic literature review to understand what constitutes a bare-minimum VR software system. The results were obsolete and no longer significant to comprehend contemporary VR software systems. Most of the primary and secondary studies conducted by Levy and Bjelland (1994) and Sherman and Craig (2003) suggested superficial information on the workings of a typical VR software system. Most secondary studies present VR from an application point of view with no precise details on the underlying aspects of VR as a domain. We found studies that explain VR as a software system applied in various fields like education, tourism, simulation, healthcare, and design applications with no underlying information about using the constructs of a bare-minimum VR software system. Given the limited academic literature, we adopted the Socio-Technical Grounded Theory (STGT) approach (Hoda, 2021) to investigate components of bare-minimum VR software systems.

STGT is a modern version of traditional sociological Grounded Theory methodology, specifically designed for software engineering

and other socio-technical domains. It is based on over 15 years of experience and combines socio-technical principles with grounded theory methods to explore the intersection of technology and society. STGT is intended to provide increased clarity and flexibility in its methodological steps and procedures (Hoda, 2021). It is an iterative and incremental research method using available resources with abductive reasoning for theory development. As per the STGT approach, the following particulars represent the boundaries of our work:

- Domain and actors: VR is our domain, and the VR practitioners who take part in VR software development are considered the actors. In this study, VR designers, developers, acoustic engineers, VFX artists, VR testers, UX engineers, release engineers, etc., are considered VR practitioners.
- Phenomenon: A meta-model for VR, illustrating a bare-minimum VR software system
- Data/tools/techniques: Qualitative data collected through informal interviews, VR SDKs, and VR standards.
- Researcher: This study is reviewed and managed by VR experts who act as researchers.

Below are the detailed steps that are required to be applied to conduct the STGT in practice:

- Define the socio-technical research context: This involves understanding the phenomenon being studied, the domain and actors involved, the researcher's role, and the nature of the collected data.
- Conduct basic data collection: Data are collected through various methods, such as observations, interviews, and document reviews, focusing on the socio-technical aspects of the research context.
- Analyze data: The collected data are analyzed using inductive and deductive reasoning, identifying themes, patterns, and concepts related to the research question.
- Develop emergent or structured theory: A theory should be developed based on the analyzed data, using either an emergent or structured approach, depending on the nature of the research question and the data collected.
- Iterate and refine: The previous steps should be repeated, refining the theory and updating the analysis as new data are collected and analyzed.
- Report findings: The findings should be documented and reported, ensuring that the research process and the developed theory are transparent and accessible to others.

Based on the STGT approach, we defined our research context as follows.

> Research context: The proprietary standards failed to create cross-platform support for VR software, driving the VR technology domain into a platform-dependent system.

We relied on the following resources to gather the data needed to establish the theory for constructing a meta-model for VR software systems: informal interviews, reviewing VR SDKs, and reviewing VR standards. These are detailed below to understand the criteria of our data collection approach.

2.2.1 Informal interviews

We conducted informal interviews with developer communities of UNITY Technologies, Epic Games, Khronos Group (OpenXR), and the VR/AR Association (UK/APAC) for nearly 6 months (October 2021 to April 2022). These interviews aimed to understand practitioners' perspectives of a bare-minimum VR software system in practice. Participants with a minimum of 5 years of VR development experience were considered for interviews. In total, 39 VR practitioners participated. The following questions were the basis for the informal interviews.

- What do you consider the desired elements of a meta-model for a VR software system?
- Do current VR development tools explain elements of a bare-minimum VR software system?
- Did you build any supporting tools for VR? If yes, how did they gather meta-information about a bare-minimum VR system?

2.2.2 VR SDK selection

Our interactions with VR practitioners led us to review widely used VR SDKs like UNITY3D (2022), Unreal Game Engine (2022), CryEngine Cry (2021), AframeJS (2022), and Amazon Lumberyard (2022). We examined SDK source code, underlying classes, and white papers. All other non-technical proprietary materials are excluded from examination.

2.2.3 VR standards' selection

Interactions with VR practitioners led us to explore VR standards in detail. We considered prior standards—VRML (W3C, 1997; X3D, 1997), WebXR, and O3D (Google Inc., 2010), and prevailing standards—OpenXR (Group, 2019) and IEEE VR/AR Working Group for our study. Additionally, we examined peer-reviewed publications to understand the components of a meta-model for a bare-minimum VR software system.

We used the open coding method (Strauss and Corbin, 1967) to annotate the interview transcripts, VR standard documentation, and VR SDK documentation with essential details. Our annotation criteria are to check for the presence of elements that explain the constructs of a bare-minimum VR software system and depict the control and data flow among them. These annotations are linked with generalized ***codes***, which have the same meaning as our annotation criteria. These codes are further generalized into common labeled ***concepts***. The researcher has the flexibility to define the concept labels. The concepts are ordered into a ***category*** goal for our study. Figure 2 explains a few examples of open codes linking overall concepts and categories to provide an overall understanding of a bare-minimum VR software system.

Based on the codes generated using open coding, we used *Abduction Reasoning* (Hoda, 2021) to theorize a meta-model for a bare minimum VR software system. Abductive reasoning helps researchers conduct data analysis through different means such as hunches, clues, metaphors or analogies, symptoms, patterns, and explanations. This approach opens various avenues for creative thinking and theory development. Following are the data points

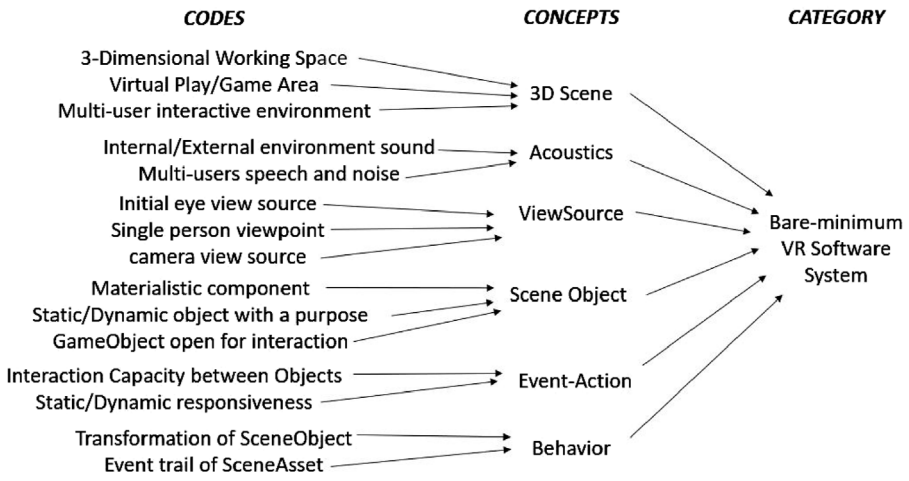

FIGURE 2
Common codes from interview study.

and observations captured to depict bare-minimum concepts that are found to constitute a VR software system. These concepts are the building blocks of a meta-model understanding of a VR software system.

- *Scene*: A 3D environment or space with (un)limited dimensions in terms of length, breadth, and height with elements operating together as a whole in their respective parts. It is referred to as a "play area" in VR SDKs and "virtual operating space" in most VR standards.
- *Article*: A 3D object with specific dimensions, state, and physical properties, including material type, texture, and color. It also has other properties like Pixel/Voxel type, CanCastShadow, IsRigidObject, IsCollidiable, CanRotate, and IsLuminous. These objects are engaged as static or interactable with the character within a given scene.
- *Action*: Any engagement between two or more articles leads to interaction, causing a known/unknown outcome. The engagement may be internal and external to objects within the prescribed scene.
- *Audio*: This element is associated with the scene and article. A scene may or may not have background audio. An article may or may not give rise to audio internally or externally due to an action by another article within the prescribed scene.
- *Behavior*: The action's outcome and the object's transformation within the prescribed scene.
- *Viewsource*: An initial viewpoint of a person trying to experience the scene. It is called a *camera* in VR SDKs and *viewpoint* in VR standards.
- *Timeline*: All the possible events are categorized into synchronous and asynchronous events that are based on a trigger, i.e., due to external stimuli. There is a possibility of non-trigger-based synchronous and asynchronous events that are within the article and will not be based on a trigger, i.e., not due to external stimuli.

The culmination of the above concepts varies between various VR standards and VR SDKs based on their levels of abstraction and nomenclature. However, the overall concepts described above represent the bare-minimum set to construct a VR software system, and they can be consistently observed across various VR standards and VR SDKs in different forms. To understand how SDKs perceive the same underlying meta-model elements differently, we present examples using WebXR APIs and AFrameJS VR scenes. Figure 3A shows a WebXR scene presenting no-audio 3D environment with a centered view-source. The scene features cube articles placed at a particular height, which change color when clicked but otherwise exhibit no interaction. Figure 3B shows an AFrameJS scene presenting a no-audio 3D environment with a centered view-source. In this scene, articles of varying sizes are placed at a distance and exhibit a change in terms of their dimension in response to any external intervention, with no actions taking place. These two scenes are different and are built using different VR SDKs. These two SDKs work under different meta-model concepts and code templates. The WebXR-based scene is obsolete as prevailing browsers no longer support this standard. The AFrameJS based scene is supported by JavaScript-based browsers only (Chrome, Firefox, etc.). These two scenes are built for the web and are not compatible with high-end head-mounted-device consumption. One of the SDKs, WebXR, is now deprecated, thus limiting the portability of all WebXR scenes and causing platform fragmentation. Such gaps can be avoided if they are built using a shared underlying meta-model of VR. Considering these observations, we provide our steps toward developing a role-based model template that represents a shared meta-model for a VR software system.

2.3 Virtual reality software system meta model

The Unified Modeling Language is a programming language with an independent notion for specifying, visualizing, constructing, and documenting systems. It is an Object Management Group (OMG) standard language for object-oriented modeling. The

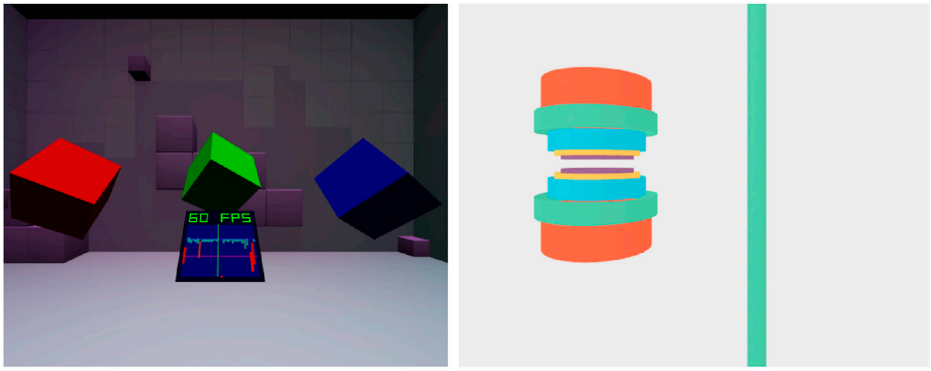

FIGURE 3
Example VR scene built using WebXR APIs and AFrameJS. **(A)** A WebXR scene with a centered-view source. **(B)** A AframeJS scene with a centered-view source.

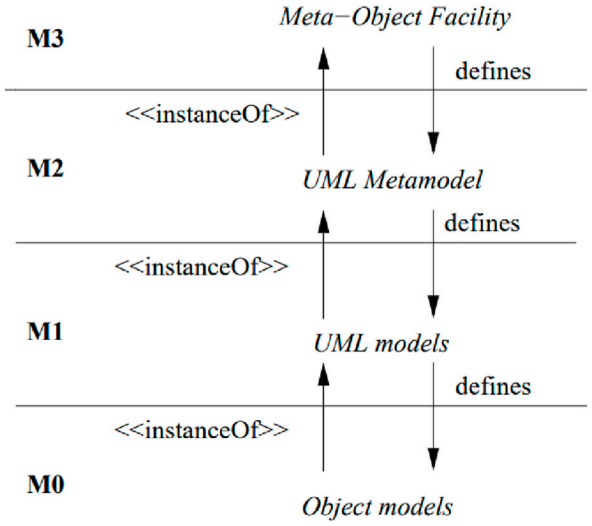

FIGURE 4
UML four-layered meta-model architecture.

UML infrastructure is defined as a four-layer architecture, as shown in Figure 4.

- Level M3 (meta–meta-model layer) defines a language for specifying meta-models. The meta-object facility is an example of a meta–meta-model.
- Level M2 (meta-model layer) contains models that specify modeling languages. The UML meta-model and the common warehouse meta-model (CWM) are examples of meta-models.
- Level M1 (model layer) contains models that describe semantic domains. The model layer consists of models expressed in languages specified by M2 meta-models.
- Level M0 (user data) consists of object configuration specified by the models at level M1.

The UML was designed primarily as a notation for modeling a single application, and its use to model application families is problematic. The role-based meta-modeling language (RBML) is a UML-based language extension that supports rigorous specification of patterns that characterize a family of design models proposed (Kim et al., 2003). As RBML uses UML syntax, UML tools can be used to create RBML specifications. A RBML specification consists of a set of role models that describe pattern properties from different perspectives with additional details. A variant of the RBML that facilitates the generation of compliant models from pattern descriptions is used in this work. Template diagrams rather than role models are used to specify families of models. The UML models are obtained from template diagrams by binding the template parameters to actual values.

A **Role** represents a specific set of responsibilities, behaviors, or functions within a system or model. It defines expected behaviors and interactions without specifying the concrete implementation. Roles are typically more abstract and focus on the purpose or function an entity fulfills in a given context. In contrast, A **Template** is a predefined structure or pattern that can be reused and instantiated multiple times. It provides a blueprint for creating consistent instances of a particular element or component. Templates are more concrete and focus on the structure and attributes of an entity. In practice, we can use both roles and templates as part of a meta-model, where *roles* help define the abstract behaviors and interactions between entities and *templates* to provide concrete implementations or structures for those roles. For example, a **Developer** role represents the responsibilities and interactions of a developer in the process, and **DeveloperTemplate** defines the specific attributes and structures of a developer entity in the system. This combination allows the model to represent both the dynamic and behavioral aspects (with roles) and the static and structural aspects (with templates) of the system. As the focus is on illustrating static and structural aspects of a VR software system, as part of our work, we use the notion of templates to describe the meta-model of the VR technology domain.

Figure 5 illustrates an example of a class diagram using RBML specifications to describe a meta-model system that authorizes a requestor by identifying the desired operation from the authorizing repository. Here, |**Requestor**, |**Authorizer**, and |**AuthRepository** are class templates. |**Requestor** carries an attribute called |reqId as

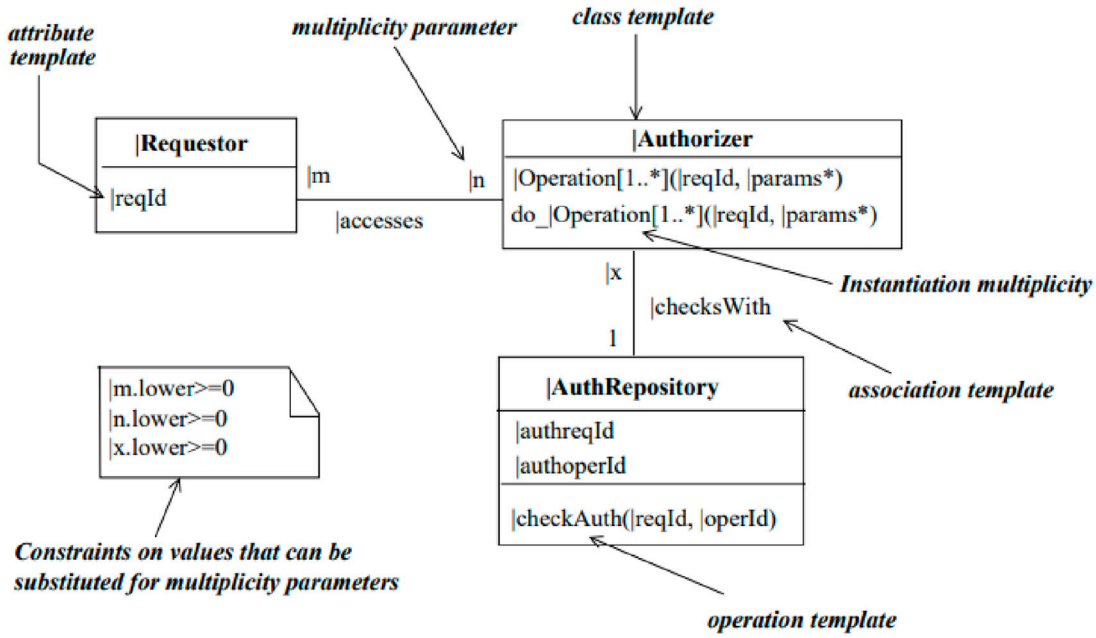

FIGURE 5
RBML class diagram of the model template for requestor authorization workflow.

part of the class attribute template. |**Authorizer** carries an attribute function called |**Operation** with parameters |**reqID** and |**params***; in other words, any additional domain specification parameter is allowed with a multiplicity of one to many along with instantiation multiplicity through **do_|Operation** as part of its attribute template. |**AuthRepository** class template carries |**authreqId** and |**authoperId** as part of the attribute template and |**checkAuth** with parameters |**reqId** and |**operId** as part of the operation template. The |**Requestor** and |**Authorizer** class templates are associated with the association template called |**accesses** with multiplicity parameters such as |**m** and |**n**, where |**m.lower** > = **0** and |**n.lower** > = **0.** |**Authorizer** and |**AuthRepository** class templates are associated with an association template called |**checksWith** with multiplicity parameters such as |**x** and **one**, where |**x.lower** > = **0.** The meta-model (M2) in Figure 5 will aid technology domain-specific software practitioners to adopt and execute the required authorization workflow in underlying applications based on their model instances (M1) to eventually develop application object models (M0).

Based on role-based meta-modeling language, we present a role-based model template to illustrate meta-model for **Virtual Reality** as the technology domain. Figure 6 presents the role-based model template class diagram of a bare-minimum VR software system. Template elements are marked with the "|" symbol. As shown in Figure 6, |***Scene***, |***Viewsource***, |***Time***, |***Behavior***, |***Physics***, |***Requestor***, |***Article***, |***Audio***, |***State***, and |***Action*** are called the class templates. Each class template consists of two sections, namely, the attribute template and the operation template. For example, the |***Audio*** class template has |***sourcetype***, |***noise***, |***init***, **and** |***inext*** as parts of the attribute template section and |***runsound***, |***syncsound***, and |***asyncsound*** are listed as parts of the association template section. Each association template is defined with a cardinality [**1..***], i.e., one to many with |***param**** as unlimited parameters. Each role model template class is linked with UML-based relationship specifications with association definitions like |***acesses***, |***Impartswith***, |***RendersInto***, |***Syncwith***, and |***Validateswith***. The virtual software system is invoked by a |***Requestor*** class template. |***Audio***, |***Action***, and |***Scene*** class templates are initiated synchronously to load underlying |***Article*(s)** with their |***State*** and respective |***Behavior*(s)** through a |***ViewSource*** onto |***Scene***. |***ViewSource*** initiates all other class templates asynchronously with |***Time***.

Figure 7 illustrates the UML four-layered meta-model architecture, deduced by considering the model template for the VR technology domain in Figure 6.

- Level M3 (meta–meta-model layer) defines a language for specifying meta models, i.e., RBML meta-object factory. It helps us illustrate class templates, attribute templates, operation templates, association templates, multiplicity parameters, and instantiation multiplicity.
- Level M2 (meta-model layer) contains models that specify VR software system meta-models. It describes the class templates, attributes, associations, and multiplicity parameters related to the VR technology domain.
- Level M1 (model layer): Using the meta-model, we create tools associated with the VR technology domain, like software development toolkits, domain specification languages, and productivity tools for the VR community to ease design, testing, coding, and release.
- Level M0 (end-user application): This layer presents the end-user applications developed using the tools developed by the VR community.

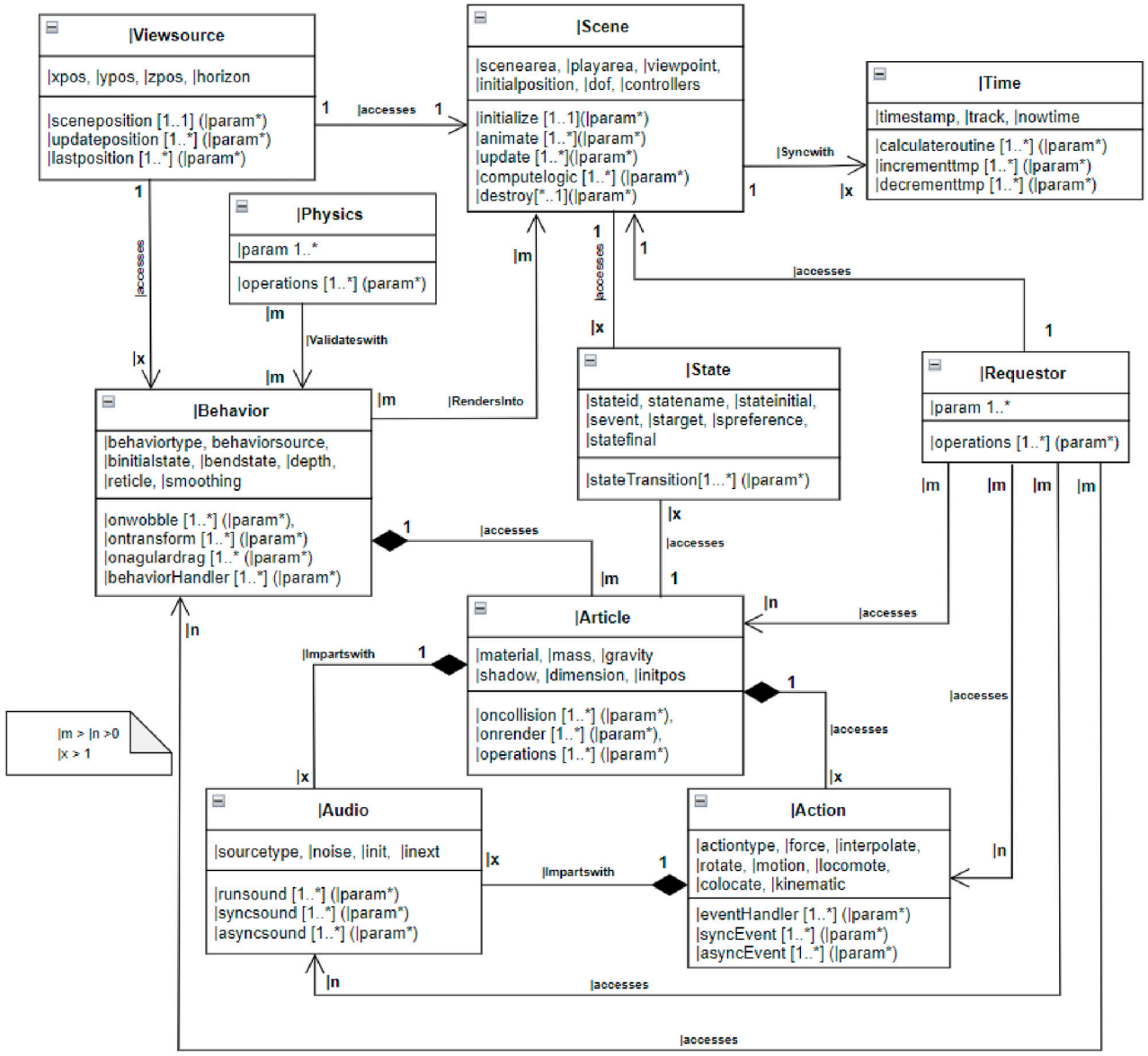

FIGURE 6
Role-based model template of a bare-minimum VR software system.

To understand the instantiation of the VR meta-model into the VR application, we bring an example that provides bindings between the meta-model and its related application using a virtual reality soccer training application developed by Rezzil Inc. Figure 8 provides us a VR scene where the soccer trainee is required to perform a task called "**Kick.**" Table 1 provides the bindings for a test case to verify the **action:kick** and the response **audio:splash** associated with **article:ball**. The example bindings can be used as part of a novel VR test case generator tool. This table provides a correlation between the meta-model attributes and potential test-case generator application-specific attributes for a VR soccer training application instance. Using the test-case application specification, a custom test-case generator tool can be developed to generate test cases for a game like VR applications.

2.4 Extending VR meta-model to domain-specific and context-specific applications

The role-based model template for a bare-minimum VR software system can be extended to an application centered on a domain or to a context within a given domain. In the context of software applications, a domain refers to the specific field or area of expertise that the software program is designed to serve. It represents the concepts, processes, and rules that define the problem space the software program aims to solve. Application domains vary widely, ranging from healthcare and finance to gaming and social media. Domains are often associated with specific industries or business processes and may involve specialized terminology and workflows. Software developers use domain models to represent these domains and create tailored software solutions to meet their needs. By

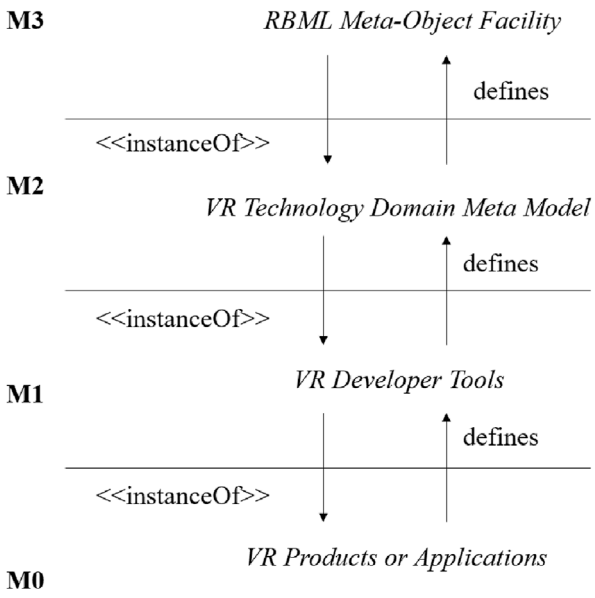

FIGURE 7
UML four-layered meta-model architecture for the virtual reality domain.

TABLE 1 Example bindings for a test-case of kicking the ball using the template class.

Meta-model parameter	Application-specific element
|Action	Kick
|Audio	Splash
|Article	Ball
|Action::|kinematic	Kick::kinematic
|Action::|motion	Kick::motion
|Action::|syncEvent	Kick::syncEvent
|Audio::|init	Splash::init
|Audio::|syncsound	Splash::syncsound
|Impartswith	impartswith
|accesses	accesses
|x	*
1	1
|m	*

understanding the domain, developers can design context-specific applications and effectively address the challenges users face in that field. As shown in Figure 9, we can visualize the top–down stack of layers using the VR model template for domain- and context-specific applications by extending underlying attributes from bare-minimum attributes to domain- and context-specific attributes.

For better illustration, let us consider that x represents the bare-minimum attributes of the VR model template called V^0, where $V^0(x)$ represents the collection of bare-minimum attributes that belong to the VR model template. Now, y represents the customized attributes with an extended role-based model template called V^1, where $V^1(y)$ represents the collection of extended attributes belonging to the extended VR model template. Thus, as per Equation 1, the collection of bare-minimum attributes from the VR model template and customized attributes together now constitutes the overall extended role-based model template.

$$V^0(x) \cup V^1(y) = V^{01}. \tag{1}$$

For example, |material is one of the bare-minimum attributes of the |Article class in the VR model template from Figure 6, which belongs to $V^0(x)$. This is a bare-minimum attribute as it is generic across all the VR developer engines. However, customized attributes like |roughness are not considered bare-minimum properties of a |material; they are unique to specific VR engines and belong to $V^1(y)$, extending the bare-minimum model template. In contrast to the domain-specific model templates V^2, these are further extended by including domain-specific attributes represented by z, where $V^2(z)$ is the collection of all such domain-specific attributes that belong to the domain-specific VR model template, as shown in Equation 2.

$$(V^0(x) \cup V^1(y)) \cup V^2(z) = V^{012}. \tag{2}$$

For example, consider the pharmacology domain in healthcare, where a domain-specific practitioner (pharmacologist) uses various chemicals to develop, identify, and test the drugs to cure, treat, and

FIGURE 8
VR soccer training application by Rezzil Inc.

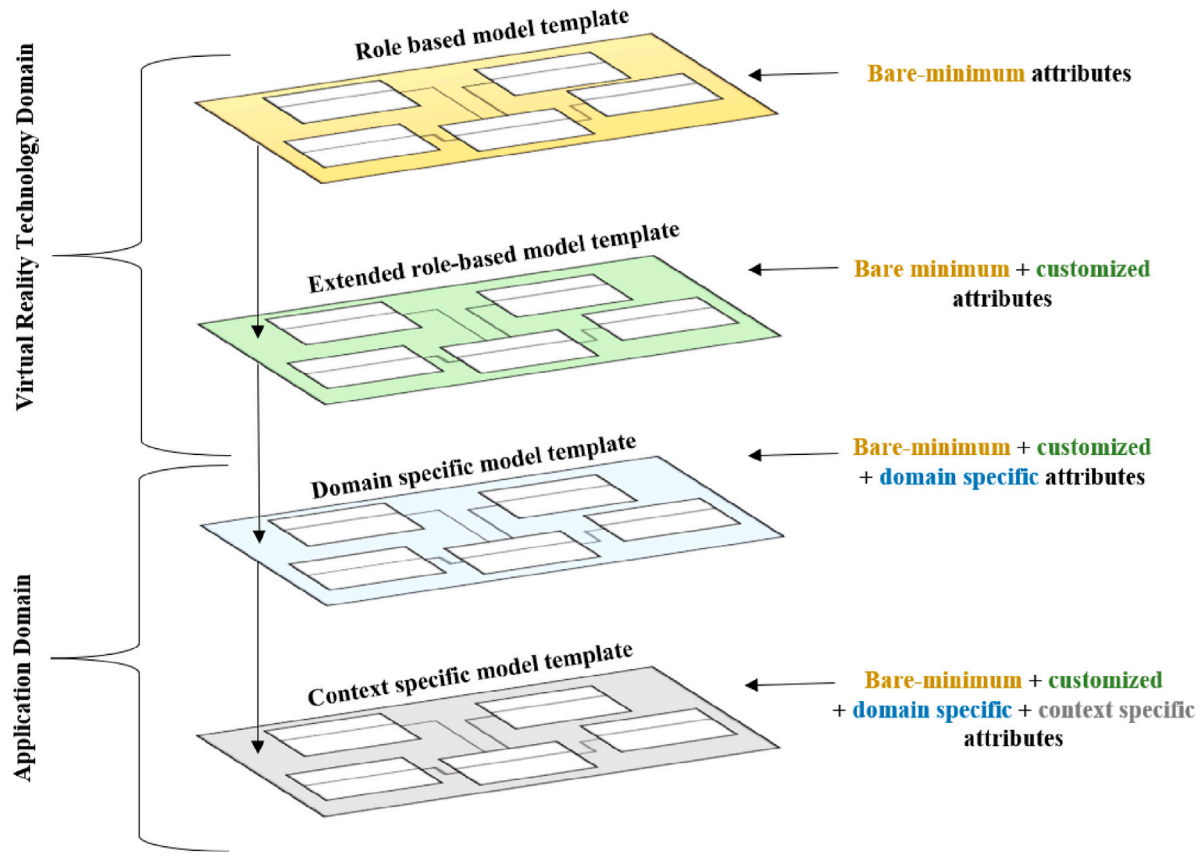

FIGURE 9
Visualizing top–down stack layers on extending the VR meta-model.

prevent diseases. A **microcentrifuge** is one of the many apparatus they rely on to spin down and separate chemical particles in small volumes of liquid samples. Consider a pharmacology domain-specific VR scene developed to train pharmacologists; a pre-curated domain-specific article, such as |**microcentrifuge,** will be equipped with |**material** properties that are unique to the pharmacology domain. Similarly, the |**microcentrifuge** can be used for different context-specific use cases, such as the molecular separation of cell organelles (like various nuclei, DNA, or RNA), phenol extraction, and studying the effects of drugs on various biological systems. Suppose a VR training scene is developed for pharmacologists for such context-specific applications V^c. In that case, VR practitioners must rely on context-specific knowledge and standard operating procedures to design the VR application. In such cases, context-specific attributes will be included as part of VR application development, which is still an extension of the domain-specific model template. As shown in Equation 3, if V^c is a context-specific application, it constitutes model attributes of V^{012}, which are from domain-specific model template and combination of various contexts, i.e., V^{c1}, V^{c2}, V^{c3} $\ldots V^{cn}$. Here, n can represent an infinite number of contexts for a given application domain.

$$V^c = V^{012} + V^{c1} + V^{c2} + \cdots . + V^{cn} \quad where \quad 0 < n < \infty . \qquad (3)$$

Figure 10 illustrates this example by presenting the model attributes in layers. This figure shows that |**material** is a bare-minimum model attribute (V^0), |**roughness** is a custom model attribute related to the VR technology domain (V^{01}), and |**microcentrifuge** is a pharmacology domain-specific model attribute (V^{012}), which is a sub-domain of the healthcare domain. |**picomicrocentrifuge** is a context-specific model attribute (V^{c1}) that belongs to a specific use case of pharmacology experiments, which inherits the properties of |**microcentrifuge** but maintains distinctive properties in practice.

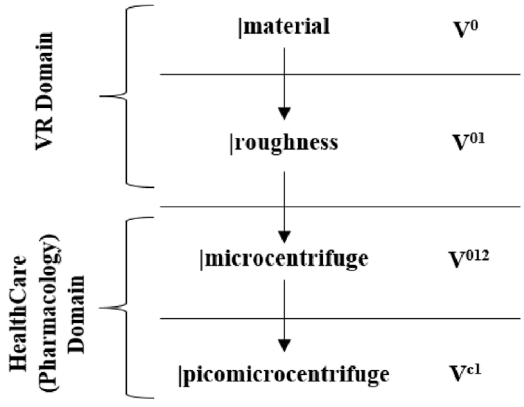

FIGURE 10
Visualizing example of extending the VR meta-model to the healthcare (Pharmacology) application domain.

FIGURE 11
Visualizing example of extending the VR meta-model to sphere game-object (these images are generated by AI). **(i)** A sphere with bare-minimum properties. **(ii)** A sphere with variations of roughness property. **(iii)** A domain specific game object, volley ball and a basket ball. **(iv)** A context specific game object, types of cricket game balls.

Thus, it is a unique context-specific model attribute within the pharmacology domain. Thus, a role-based model template for a VR software system can be extended to a customizable, VR technology domain-centered model template. It can be extended further to a domain-specific application and a context-specific applications within a given application domain.

Figure 11 illustrates an example of a sphere game object. In Figure 11i, the sphere shows bare-minimum properties, including dimension, color, and illumination. In Figure 11ii, the sphere shows variations in the roughness property, which is not a bare-minimum attribute. In Figure 11iii, the sphere illustrates a domain-specific game object, such as volley ball and basketball, with variations in both sphere dimension and roughness. In Figure 11iv, a context-specific game object is shown, representing various balls used in the game of cricket. Thus, a role-based model template for a VR software system can be extended into a customizable, VR technology domain-centered model template. It can be extended further to a domain-specific application and a context-specific application within a given application domain.

2.5 Model-based requirement specification for virtual reality software

2.5.1 VR authoring tools

An authoring tool is a software application or platform that enables users to create content in a cohesive and interactive format, including text, code, graphics, audio, and video. These tools simplify content creation, making it accessible even to those without advanced technical skills. In the context of the virtual reality technology domain, VR authoring tools are explicitly created to develop interactive and immersive virtual reality content. They enable users to develop VR experiences without requiring extensive programming knowledge. They offer an intuitive user interface, integration of multimedia assets, support for a wide range of VR devices, collaborative features across various VR developer engines, and integration support to external software like learning management systems to host VR content. Sometimes, VR authoring tools cater to different expertise levels, offering pre-made interactions and scenes for non-specialists or customizable content for specialists. Unfortunately, most VR authoring tools do not incorporate authoring capabilities as an integral feature in the VR developer tools. Due to serious platform fragmentation issues within the VR technology domain, individual VR practitioners develop custom plugins to streamline the VR content to modify and distribute across distinct VR developer engines. Examples of VR authoring tools include ***AirVu Authoring Tool***, ***LearnBrite***, ***Storyflow***, ***and CenarioVR***. These tools cater to various industries, such as education, healthcare, manufacturing, and customer service, providing customizable solutions predominantly for developing immersive VR training modules. It is pretty rare to observe a VR authoring tool that conventional VR practitioners can use as a general-purpose tool. This is primarily because the artifacts generated from such VR authoring tools may not work across different VR developer engines and do not support multiple VR devices.

2.5.2 VR requirement specification tool

Requirement specification tools help manage, organize, and document software requirements throughout the development of software products. Their primary goal is to ensure that all stakeholders involved in a software project have a clear understanding of the requirements and that the final product aligns with the initial vision. Such tools are required to support the requirement creation, provide traceability, enable collaboration, manage changes, prioritize requirements, and offer version control of requirements, among other features. In the context of the virtual reality technology domain, there is no dedicated requirement specification tool that meets these domain-specific needs. Due to the lack of VR-centered requirement specification tools, the VR community is heavily relying on conventional approaches like software requirement specification templates and task tree-based requirement specification templates.

As shown in Table 2, we illustrate the differences between authoring and requirement specification tools. In brief, authoring tools are for authoring and organizing content based on requirements, while requirement specification tools are for managing and tracking requirements throughout development. Furthermore, the authoring tools focus on clarity and conciseness, while requirement specification tools provide a more comprehensive approach to the management of requirements. As there is a need for such comprehension in the VR technology domain, we developed a model-based requirement specification tool. In the following subsection, we provide more details about

TABLE 2 Comparison between authoring tools and requirement specification tools.

Feature	Authoring tools	Requirement specification tools
Focus	Implement the requirements into design, mockups, or workflows	Specify, track, and maintain requirements
Target audience	Designers, developers, and subject matter experts	Project managers, technical stakeholders, and customers
Key functions	Structured workflows, attribute assignment, and requirement hierarchy	Traceability, change management, version control, and specification
Use cases	Early stage of design and development	Throughout the development process
VR tool examples	AirVu Authoring Tool, LearnBrite, Storyflow, and CenarioVR	Task tree templates and conventional SRS templates

our novel requirement specification tool that can help requirement analysts specify requirements with higher precision and clarity.

2.5.3 Why use a model-based requirement specification tool for VR?

From our previous systematic literature review (Karre et al., 2024) on understanding requirement engineering methods for VR products in practice, we observed that most VR practitioners relied on approaches like conventional functional systematic specification documentation, common scene definition framework, and mental model techniques to specify requirements for VR products. We observed from our literature review (Karre et al., 2024) that due to a lack of understanding about VR as a domain within the VR community, novel tools are not practiced or adopted. Thus, we envisioned developing a tool that works on the foundations of a meta-model of a VR technology domain. The practical benefits of model-based tools (Gonzalez-Perez and Henderson-Sellers, 2008) motivated us to build a model-based requirement specification tool for VR software. These benefits are as follows:

- They reduce ambiguity and misinterpretation compared to natural language specifications
- They provide better traceability between different levels of requirements and system components, making it easier to track how changes in one area impact others.
- Models can facilitate better communication between stakeholders, including non-technical team members.
- The elements and constraints of the model can be more readily validated against stakeholder needs and system constraints, often through simulation or formal verification techniques.
- Model elements can often be reused across projects, potentially saving time and effort in future development efforts.
- As systems become more complex, model-based approaches can help manage and analyze intricate relationships and behaviors that might be difficult to capture with traditional text-based specifications (Arshad et al., 2023).

2.6 VReqST—to specify VR requirements

The **VReqST**, i.e., "*virtual reality requirement specification tool*," is developed for requirement analysts of the VR community to specify requirements using a role-based model template of the VR software system (Karre et al., 2022). VReqST will aid requirement analysts in describing scene properties, article properties, and action responses between these articles in the given scene, along with their state change and custom behaviors (like algorithms and logic interpretation) that are to be executed in a defined timeline. Our first iteration for VReqST was oriented toward a formal-specification language with pre-defined code constructs. We exchanged our early ideas with requirement analysts from the VR community through various platforms. Based on the feedback from the VR community, we simplified them into a model template tool instead of evolving it into a formal-specification language. In contrast, VReqST behaves as an informal form of a specification language. In the following sub-sections, we present the details about different model template(s), their underlying validator(s), the architecture, tool overview, and workflow of VReqST.

2.7 Model template and its validator

As mentioned in the previous section, model templates are extensions of model concepts defined as part of the role-based model template for VR software systems (Karre et al., 2022). These model concepts are scenes, articles, action responses, behaviors, and timelines. Each model concept constitutes model attributes that act as properties of the respective model concept. Table 3 provides the mapping of model concepts and their model attributes. In the case of VReqST, we represent each model concept as a model template file. They are represented in the JavaScript Object Notation (JSON) format. The model template file contains model attributes, a minimum set required to capture requirements for VR software products, as shown in Table 3. For example, the model template file of the model concept called "**Article**" contains one of the model attributes called "*IsKinematic*"—a physics property of the article that can execute motion when an external force is applied. The scope of this attribute is Boolean, i.e., it can only hold either value "*1*" or "*0*," where "*1*" signifies that the respective article holds kinematic physics property, whereas "*0*" signifies that the respective article does not hold kinematic physics property.

All such validation rules for model attributes for a given model template are defined as part of the underlying validator. Each model template file (scene, article, action response, and timeline) has a distinct validator file to validate the datatype and scope of the model attributes, as shown in Figure 13. These validator files are also

TABLE 3 VReqST model concepts with corresponding model attributes.

Model concepts	Model template file	Validator file	Model attributes
Scene	scene.json	svalid.json	_scenename, _sid,_slabel, _playarea, #pid, #length_playarea, #breadth_playarea, #height_playarea, #comment, #x_scenecenter, #y_scenecenter, #z_scenecenter, _camera, IsSceneObject, trackingorigin, _initialcamerapos, _viewport, _clippingplane, _horizon, _dof, _skybox, _controllers, _gravity, _interaction, _nestedscene, _audio, _timeline, _Opttxt1,usertype, uplayarea, initialupos, uplayareacenter
Article	article.json	artvalid.json	_objectname, _sid,_slabel, _IsHidden, _enumcount, _Is3DObject, HasChild, shape, dimension, IsText, IsText3D, CastShadow, ReceiveShadow, ContributeGlobalIllumination, IsIlluminate, Transform_initialpos, Transform_initialrotation, Transform_objectscale, repeattransfrom, XRGrabInteractable, XRInteractionMaskLayer, TrackPosition,TrackRotation, Throw_Detach, forcegravity,velocity, angularvelocity, Smoothing_duration, attachtransform, hasmass, dragfriction, angulardrag, Isgravityenable, IsKinematic, CanInterpolate, CollisionPolling, aud_hasaudio, aud_type, aud_src, aud_volume, aud_PlayInloop, aud_IsSurround, aud_Dopplerlevel, aud_spread, aud_mindist, aud_maxdist,_Opttxt1, @context_img_source
Action response	actres.json	arvalid.json	actresid, sourceObj, targetObj, IsCollision, response, comment, Syncronous, repeatactionfor
Timeline	timeline.json	tvalid.json	animate_trigSync, tsyncid, tsOntriggertrue, SyncObjList, tSyncNote, animate_nontrigSync, ntsyncid,ntsOntriggerfalse, ntSyncObjList, ntSyncNote, animate_trigAsync, tasyncid, taOntriggertrue, AsyncObjList, tAsyncNote, animate_nontrigAsync, ntasyncid, ntaOntriggerfalse, ntAsyncObjList, ntAsyncNote, routine, routeid, starttime, endtime, order

represented in the JSON format. The model template files and their corresponding validator files are available as part of our documentation (Karre, 2024a). Table 3 provides details of the default naming convention for the model template file and validator files for the respective model concept. The requirement analysts can rely on VReqST documentation to specify requirements in the model template file while eliciting requirements from their stakeholders. These model template files will aid requirement analysts in probing the requirements more accurately.

2.8 VReqST overview

VReqST is a web-based tool built using the Mongo–ExpressJS–ReactJS–NodeJS (MERN) technology stack, i.e., a ReactJS framework is for client-end web page rendering, ExpressJS framework and NodeJS for server-end setup, and MongoDB as a database to store the data. Figure 12 illustrates the architecture of the VReqST tool. The source code, tool demo, documentation, and sample requirement specification examples are available as part of our resources (Karre, 2024a). Requirement analysts of VR products are the target users of VReqST. This web-based tool is designed to aid requirement analysts in capturing requirements instantly and refer to them effortlessly. This will eventually ease the work of VR designers and developers by streamlining the development of VR software products and enabling swift traceability to the original requirements.

As shown in Figure 13, there are two sections in the VReqST tool, namely, authoring wizard and validator engine. Requirement analysts interact only with the authoring wizard while specifying VR software product specifications, whereas the validator engine runs in the background. As shown in Figure 13, the requirement analysts are required to start with scene-related details, i.e., the scene model template is the first stage of the VR requirement specification. Once the required model attributes of the scene model template are composed, the specification is validated using the scene validator. Upon successful validation, the requirement analyst can compose an article model template, a second stage of the VR requirement specification. All the dependent scene model attributes are carried forward to the article model template. The article model attributes are validated using the article validator. Upon successful validation, scene- and article-related model attributes are carried forward to the following stage, i.e., action response, the third stage of VR

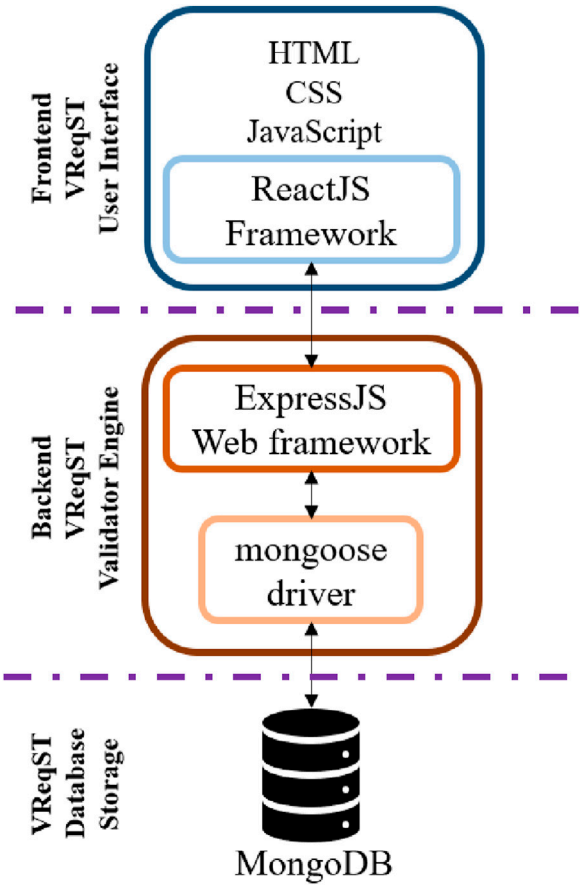

FIGURE 12
VReqST architecture.

requirement specifications. The requirement analyst must define all possible action responses between articles that are listed as part of the article model template in the second stage. Along with all possible action responses, the state of an article and its initial, final, and transition states are to be defined. After completing the action response model template, all the scene, article, and action response attributes are carried into the customer behavior editor. This is the fourth stage of authoring VR requirement specifications. The custom behavior editor in VReqST will help requirement analysts define behaviors using logic and conditional constructs. For example, in the case of VR games, scoring criteria can be codified as part of this stage. In the case of VR simulation scenes, custom algorithms like co-location or locomotion can be codified. The codified code constructs are validated based on the behavior validator. After authoring behaviors, the requirement analysts are required to present the overall specifications in a timeline. This is the final stage of the VR requirement specification. The attributes defined in all the previous stages can be presented as asynchronous and synchronous events. A timeline validator validates the attributes associated with the timeline model template. After completing all the stages, a final specification file combining all the model attributes will be published. This final specification is now available for designers and developers to initiate VR software product development. All the stages in VReqST are linear and cannot be skipped, i.e., the specification will start with a scene, article, action response, behavior, and timeline. We may not be able to specify articles directly without specifying scene information. We can navigate back to the respective stage if validation of a given stage is complete. We shall go through the detailed validation workflow of VReqST in the following subsection.

2.9 VReqST validation workflow

Figure 14 illustrates an overall validation workflow of specifications of the model template using the respective validator. This section is divided into two parts: validation of model template files and validation of the custom behavior editor.

2.9.1 Model template validation

Figure 14A provides detailed steps for validating the model template file. There are four model templates in VReqST. This applies to the scene, article, action response, and timeline model template files; the steps for validating the model template file are as follows.

- Step 1: The requirement analyst accesses the authoring wizard and chooses a fillable model template (scene, article, action response, or timeline). This fillable model template contains all the model attributes. The requirement analyst must choose the validate option upon filling in all the required details for model attributes in a given model template file.
- Step 2: The validate button will convert the filled model template file into an abstract syntax tree with all the model attributes. The validator file contains the validation rules and validates the model attributes in the abstract syntax tree.
- Step 3: If the validation is successful, the final model template is processed by the specification processor, where all the previous step's model attributes are carried forward and linked to support traceability. The finalized specifications are presented to the requirement analyst. If the validation fails, model attributes are to be revised per the validator logic in the authoring wizard.

2.9.2 Custom behavior editor validation

Figure 14B provides detailed steps for validating the custom behavior editor. As shown in Figure 13, stages 1, 2, 3, and 5 are fillable model template files validated against respective validator files. However, stage 4 in the authoring wizard contains a custom behavior editor that allows requirement analysts to write custom behaviors on articles and their action responses. These behaviors are business rules with expected output for the corresponding output. These business rules are use-case-based and vary from product to product. The steps involved in validating the behaviors are stated as follows.

- Step 1: The requirement analyst uses the WRITE wizard to describe the behaviors using inbuilt code constructs like IF, IF-ELSE, SWITCH, FOR, WHILE, DO, and DO-WHILE. The

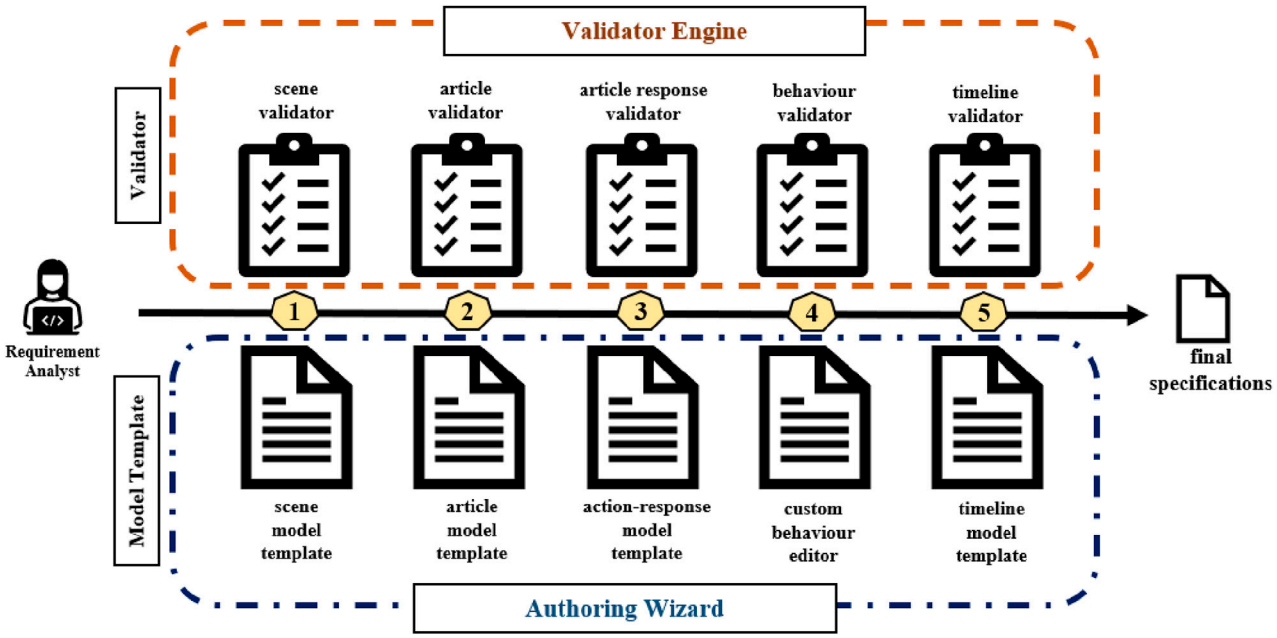

FIGURE 13
Overview of VReqST author journey.

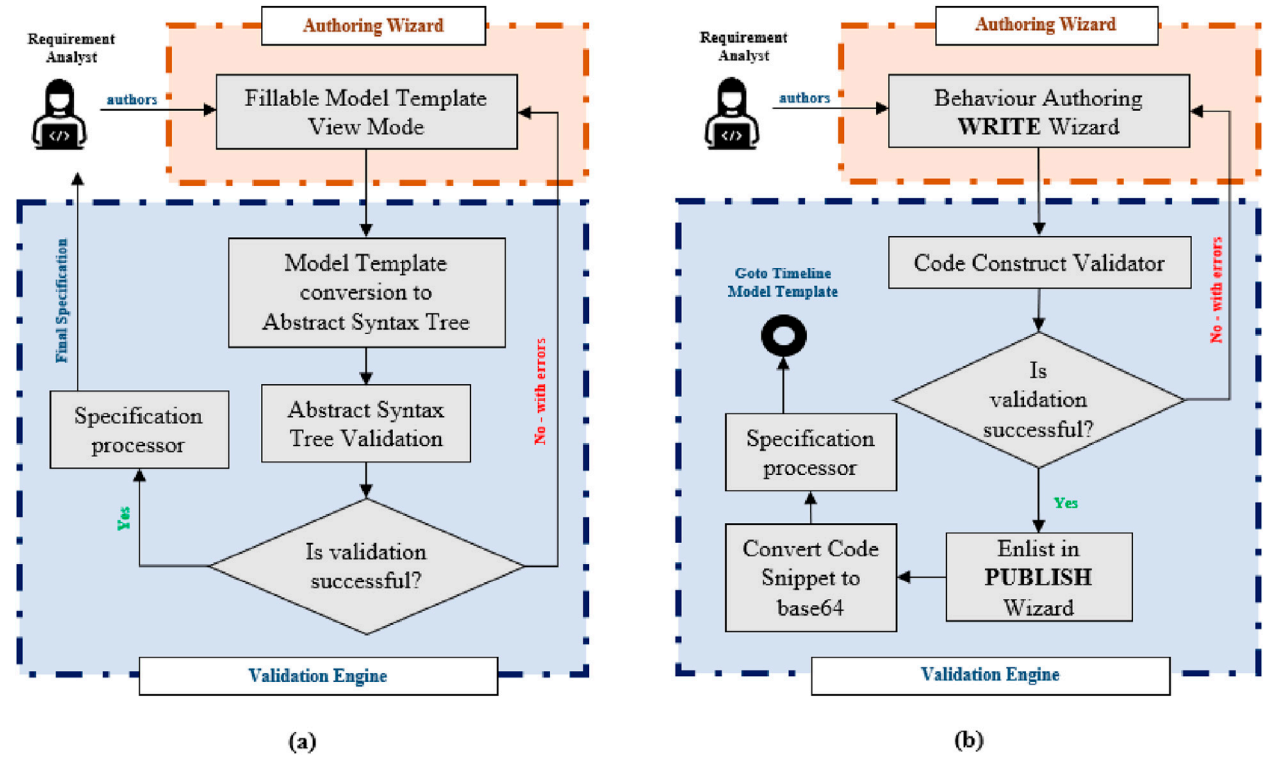

FIGURE 14
(A) Validation workflow of the model template in VReqST tool, **(B)** Customer behavior editor in VReqST tool.

syntax of these code constructs does not follow any specific programming language but follows a simplified version, as prescribed in our documentation (Karre, 2024a).

The scene, article, and action response model attributes are made available as part of the WRITE wizard to specify behaviors.

- Step 2: If the behavior logic is specified according to the prescribed code construct, the behavior logic is valid. The valid behavior logic is listed in the READ and PUBLISH wizard of the custom behavior editor. The READ wizard will help requirement analysts review the written behavior logic. Out of all the specified behaviors, the requirement analyst can pick and choose the desired behaviors to be included in the final specification. The PUBLISH wizard will help achieve this task. If the behavior logic is invalid, it is required to be specified again as per the prescribed code constructs defined as part of our documentation (Karre, 2024a).
- Step 3: The published behaviors are converted into base-64 as multi-line code snippets cannot be stored in JSON object files. All the chosen behaviors from the PUBLISH wizard are ordered under a single behavior file and are stored in the database.
- Step 4: All the previous step's model attributes are carried forward and linked with the published behavior file to support traceability. This step is carried forward into the timeline model template, which is the final step of the VReqST tool.

2.10 Understanding behavior and state in VReqST

For example, to author "**Behavior**," assume that we are building a tic-tac-toe game in VR with two articles, circle (O) and cross (X). If three of any one of these articles form a straight line (vertical, horizontal, and diagonal) in a tic-tac-toe game, the player with the respective article is deemed the winner. To author this behavior in stage 4, we use the code snippet below using a **CASE** condition, and everything in **bold** is the pre-defined code construct from *behave.json*. The text in *italics* are articles from *article.json*, and the rest is in the free-form text, which is not validated. The code snippet construct in bold is validated using behave.json.

1: **CASE #** *Circle* OR *Cross* = true **#**
2: **C1:** forms straight vertical line = *player*:*winner*;
3: **C2**: forms straight horizontal line = *player*:*winner*;
4: **C3:** forms straight diagonal line = *player*:*winner*;
5: **C4:** *player*: *next_step*;
6: **!**

The requirement analysts can specify behaviors using in-built code constructs. These code constructs do not follow existing programming standards but are custom-made for VReqST. They follow an inbuilt syntax that is simplified and easy for a non-developer to use. After considering the feedback from the VR community and considering that requirement analysts are full-scale developers, we designed them as low-code tools.

Similarly, "**State**" transition can be specified by defining state parameters like initial and final states based on transition parameters like event and target. For example, to describe the state transition of sun-rise in VReqST, the requirement is to specify that while the sun rises, the sun rays should cast a shadow and transition snow to water in a specified time counter.

Listing 1 Example: State template.

```
{
  "state": [
    {
    "id": "sunrise2013",
    "name": "Sun Rise in New York",
    "initial": "dark horizon",
    "transition": [
    {
      "object": ["Sun","trees"],
      "event": "rising",
      "target": "sun ray cast shadow on all
assests from east to west"
    },{
      "object": ["Sun","Snow"],
      "event": "rising",
      "target": "turn the snow into water
gradually after 35 seconds"
    }
    ],
    "final": "Display sunrise with cast shadow
on trees and snow converted to water
after 50 seconds"
    }
  ]
}
```

2.11 VReqST features

Following are a few significant features of the VReqST tool that are provisioned for requirement analysts to embrace based on the sophistication of their VR software product:

- Bare minimum model template and attributes: VReqST provides bare minimum model attributes related to specific model concepts by default. These default model attributes are illustrated in Table 3. The underlying validator file for the respective model template file constitutes validation rules associated with these bare-minimum attributes. The requirement analysts can use these default model template attributes to author simple VR software products.
- Implement new model template and attributes: The requirement analysts can include new model attributes without altering the bare-minimum model attributes for a given model concept. For every newly implemented model attribute in the model template file, a validation logic should exist in its validator file. For example, if a new attribute called "**color**" [inspired by OpenXR standard specification (Group, 2019)—**XrColor4f** structure] is added to *scene.json*, its validation rule for variables (*r, g, b, a*) with datatype as (*float, float, float, float*) and *non-nullable* should be included as part of *svalid.json*. Thus, supporting new model attributes extends the scope of capturing requirement specifications for VR. The steps to update these files are available as part of our documentation (Karre, 2024a).
- Managing specifications: The step-by-step process of managing projects in VReqST and detailed screen-flow are

available in supporting documentation (Karre, 2024a). Requirement analysts can rely on this document to manage their VR requirement specification projects. Requirement analysts can author requirement specifications for multiple VR products through a project management page. Each project follows the specification workflow mentioned as part of Figures 13, 14. Requirement analysts can write, save, edit, re-edit, and delete the authored requirement specifications of their respective VR products.

2.11.1 Example specifications

We used VReqST to specify requirements for two VR applications to understand the capabilities and shortcomings. We specified detailed requirements for a VR bowling alley game and a VR virtual art gallery that uses a limitless locomotion algorithm called PragPal (Mittal et al., 2022). We developed these VR scenes using the UNITY game engine (version 2022.2.16), compatible with the Oculus Meta Quest 2 head-mounted device.

2.11.2 VR bowling alley game

A VR bowling alley game is a virtual facility where bowling is played. It is a multiplayer game that follows all the bowling alley rules. The game scene is played under a 30 ft (length) × 30 ft (breadth) × 30 ft (height) virtual play area. The game contains a bowling ball, ten pins, a pinsetter for setting the pins in a frame, an alley lane, and a gutter that acts as a boundary for the alley lane. The game is controlled by a control desk that registers the parties, registers bowlers, and manages the party assignment to the game lanes. There are ten games for each party player. The scores are displayed on the scoreboard. The player from a party who scores the highest in all ten games wins. We authored and shared the VR bowling alley game specifications with the VR developer to design and develop the game. The final working game (Kandhari, 2023) and the sample specifications are available as part of our resources (Karre, 2024b).

2.11.3 VR virtual art gallery

The VR virtual art gallery is an endless corridor comprising two walls running parallel to each other. The end users walk through the gallery to explore the art exhibits on these walls and progress in the forward direction. Under the influence of an underlying locomotion algorithm, the users' path becomes virtually limitless in a limited physical space; in other words, within a fixed physical play area, the user can navigate freely in a virtual play area. We authored the requirement specifications of this virtual art gallery and the underlying limitless locomotion algorithm using the VReqST to study the capabilities of specifying algorithms using the behavior step. The final working game (Mittal et al., 2022) and the sample specifications are available as part of our resources (Karre, 2024b). This validation helped us conduct a unit test of VReqST and demonstrate its capabilities of specifying custom features and algorithms for a given VR scene.

3 Results

3.1 Validating VReqST in practice

How do we validate a requirement specification method? Software requirements are not static and evolve throughout the software development lifecycle. As the project progresses, new requirements may arise due to changing user needs, technological advancements, and stakeholder feedback. As a result, software development teams must be agile and adaptable in managing and accommodating these evolving requirements. Conversely, requirement specifications are vital in ensuring that software products meet the desired functionalities, performance, and quality standards during this evolution process. Requirement specifications serve as a guiding framework for the software development team during requirement evolution. These specifications comprehend the software's intended behavior and enable the software practitioners to make informed design decisions and develop solutions that align with the specified requirements. By adhering to these specifications, software development teams can create reliable, efficient products that meet their users' expectations. The consistency, completeness, and correctness of requirement **(R)** specification **(S)** can only be validated and verified under the context of a given domain **(D)** (Gervasi and Nuseibeh, 2000). As shown in Equation 4, given the assumption that the machine will perform as instructed by the specification and that our model of the domain faithfully predicts how the real world will behave (Gervasi and Nuseibeh, 2000),

$$S \cup D \in R. \tag{4}$$

These observations holds true even for VR software systems and their VR practitioner community. However, the critical difference is that the evolution of VR requirements is very difficult to track and supervise unless there is a structured approach to comprehending VR as a technology domain. VReqST aims to address this gap. VReqST is developed to specify, maintain, track, and manage the evolution of VR software product requirements more precisely. Considering this as our motivation, we reached out to the VR community to understand how VReqST can disrupt this evolution and aid the VR community in easing overall VR product development. In the following sub-sections, we present our attempts to validate the adoption and impact of VReqST in practice.

3.2 VR community adoption and feedback

We reached out to the VR community from the industry through various channels, like LinkedIn, VR/AR meetups, online VR summits, Discord XR Connect, the Khronos Group, Deloitte Digital, and UNITY Unite Expo (2022 and 2023), to promote VReqST for validation, adoption, and feedback. We clearly defined our evaluation study's objectives, ensuring they are specific, measurable, achievable, relevant, and time-bound. This ensures we receive a clear direction for our validation and helps incrementally improve the tool for wider adoption. We reached more than 500 practitioners from the VR community, of which only 101 participants have shown to deploy the tool in-house for validation. Of these participants, 53 have actively participated in iterations and helped us revise and improve the tool over time. These participants include business analysts, VR product managers, VR program managers, VR quality engineers, and VR developers. This multi-year empirical study was conducted between October 2022 and January 2024. The steps followed by the practitioners who have shown interest in implementing and validating VReqST in practice are as follows.

- First, we shared the official VReqST documentation and installation steps to help participants integrate the tool into their regular processes and understand its intended use (Karre, 2024a).
- We have made every effort to deploy VReqST in a way that integrates seamlessly with their existing processes and scales effectively with their existing requirement specification practices.
- We helped the participants understand the practicality and viability of implementing the VReqST tool and assess the potential benefits in relation to the costs, resources, and time required for integration with their conventional approaches.
- We requested to involve diverse teams, domain experts, and stakeholders from their internal organizations to gather varied perspectives and insights. We leveraged collective expertise to comprehend the tool's potential impact across various aspects of their VR product development. These diverse viewpoints will help us identify the tool's opportunities, threats, and unforeseen implications in practice.
- We received enhancements to VReqST and incorporated them in a new version of the tool. We received enhancements in three iterations, and we included the revisions and released a new version in all three iterations for further adoption.
- In the end, we conducted a short survey to understand the impact of VReqST in their day-to-day business when compared to their conventional practices.

In the following subsection, we present a detailed overview of our study setup, feedback received in three iterations, and the impact of survey details.

3.3 VReqST validation study setup

The following steps are involved in our validation study setup with VR community practitioners in respective iterations.

3.3.1 Iteration 1

- In early October 2022, we provided the source code of our first version, i.e., iteration 1 of the VReqST tool to deploy.
- We requested the participants to author at least 2–3 new requirement specifications of their existing projects using VReqST and share them with their VR developers.
- As part of the early analysis, we requested the requirement analysts to review the capabilities of the VReqST tool compared with their conventional requirement specification processes.
- A total of 53 unique VR practitioners have deployed the tool during this iteration.

3.3.2 Iteration 2

- We re-connected with the VR practitioners and sought their feedback in mid-April 2023. We implemented the feedback from iteration 1 and shared the revised VReqST tool for redeployment in May 2023.
- We requested the practitioners to re-author the old specifications using the revised tool and the new requirements to experience the revisions and ease of use with new enhancements.
- During this period, we collected feedback from these VR practitioners on the iteration 2 version of the VReqST tool and documented it for further improvement.
- A total of 48 out of 53 unique VR practitioners have provided feedback on the iteration 2 version of the VReqST tool.
- However, all the 53 unique VR practitioners have deployed the iteration 2 edition of the VReqST tool.

3.3.3 Iteration 3

- We received significantly valuable feedback during our iteration 2, which helped enhance the tool between July 2023 and October 2023. We completed our iteration 3 version of the VReqST tool by the end of October 2023 and delivered it to the engaged VR practitioners in early November 2023.
- All 53 unique VR practitioners have deployed the iteration 3 edition of the VReqST tool. This was the final iteration of our VReqST enhancements that significantly improved the workflow and ease of use of the VReqST in practice.

In the following sub-section, we present detailed observations shared by VR practitioners during each iteration, along with the implementation information related to those enhancements in detail.

3.4 VReqST impact and experience survey

After a thorough review of iterations and implementing feedback on VReqST, we reached out to the VR practitioners to share their experiences on using VReqST and the impact it has made on their day-to-day activities. The following are the questions posed to VR practitioners to help us understand their experiences and the impact of VReqST on their VR product development journey in contrast to their regular processes.

1. Is the VReqST tool compatible with the existing tools that you use for your overall VR software product development?
2. Is the overall user-interface and design of the VReqST tool intuitive and user-friendly for your team members?
3. Does the VReqST tool allow you to specify requirements for necessary features in your VR software to meet your business needs?
4. With whom and how frequently did you use the VReqST tool while specifying or explaining the requirements to your target users (developers or stakeholders)?
5. Are you able to completely specify and track the requirements associated with your feature in a given VR scene?
6. Does the tool allow you to correctly specify the requirement as per the conventional standards of the VR technology domain?
7. Is the VReqST tool's performance and stability under different conditions and workloads acceptable for your day-to-day business?
8. Do you believe the VReqST tool is extendable to meet your current product needs and adaptable to future needs, considering updates in the VR technology domain?

Overall, our validation study and impact and experience survey of the VReqST tool focuses on the following ***quality attributes*** listed

below in particular stack order. We evaluated the benefits of VReqST using these attributes. We share detailed observations from the VR practitioners using these quality attributes for better quantification.

- Specify: ability to define the requirements using the existing model templates.
- Customization: ability to customize and alter the model template and underlying validator to meet custom business requirements.
- Scalability: ability to scale VReqST to include new model attributes for domain- and context-specific applications.
- Traceability: ability to trace the requirements across various stages of specifications.
- Usability: ability to use the overall tool and manage track changes to the requirements.

4 Discussion

4.1 Observations—validation study

While releasing the incremental version of VReqST during three iterations, we gathered feedback through informal interviews with the VR practitioners on VReqST. We summarize the overall observations gathered during all the iterations using the "***Abductive Reasoning***" approach and illustrate them below. We categorized these observations into two parts: **merits** and **enhancements**.

4.1.1 Merits

- Meets the intention: The VReqST tool can meet its required intention, i.e., specifying VR requirements, customizing the VR model template and its validator, and scaling the VR model template to new model attributes.
- Supports traceability: The VReqST tool supports traceability. We can trace defined articles and their action responses and state information across the trail of overall specifications. This helps a VR developer and quality engineer to trace, define, and test the flow of events while developing the VR scene.
- Improves clarity: The VReqST tool helps improve requirement gaps and present clarity as the specifications are precise. It also reduces the time required to comprehend the overall implication of a specification as the model template provides details from scene to timeline.
- Enhances productivity: The VReqST tool avoids back-and-forth communication between developers and product managers on requirement clarification, eventually helping accomplish developer tasks more effectively and increasing output by saving time and effort.
- Increases adaptability: Managing requirements of various projects through project view helps requirement analysts to adapt to new workflows and achieve better results.
- Enhances collaboration: As the specifications can be shared across various projects, multiple stakeholders can collaborate on projects simultaneously, fostering teamwork and communication.
- UI/UX: The practitioners have reported 20+ recommendations for the overall UI/UX design. We included the updates in an iterative release. As the VReqST tool is still a prototype, we can extend industry-scale UI/UX features in the user interface that can impact the user experience. End-to-end usability testing on VReqST helped us improve the usability aspects of the tool.
- Reusability: The specifications authored in VReqST are either localized to a specific project or can be extended to a given new scene. The tool can save, share, and re-use the authored specifications, especially for articles and their action responses, as a repository. Such a repository bank can help other requirement analysts adopt and reuse based on their business needs.

4.1.2 Enhancements

- Simplify authoring: Currently, the requirement analysts (RAs) are required to specify requirements using the JSON model template files. If VReqST handles the conversion from form-based model templates to JSON at all stages, it may lead to higher adoption of VReqST from VR practitioners with non-programming backgrounds.
- Security and privacy aspects: Security and privacy requirements are a few key areas of overall VR software. The current version of the VReqST does not provide any model attributes linked to security and privacy, which are essential for a bare-minimum set of model attributes. This may lead to setbacks in the practice of adopting VReqST.
- Complexity: Authoring complex behavior requires much conceptual effort compared to conventional requirement specifications that are authored in plain English. The behavior editor needs to be either gamified or allowed to be in a low-code mode to improve the overall adoption of the behavior editor.

In contrast to the consolidated observations, we present detailed feedback that we gathered during each iteration below, along with the revisions performed on the VReqST tool. We shared the revised tool during each iteration and have incorporated these changes accordingly.

Iteration 1: The detailed changes conducted as part of our iteration 1 are listed below. We included all the required changes in the form of figures as part of Supplementary Material.

- Multi-stage authoring: Our initial VReqST tool did not support multi-stage authoring of our requirement specifications for available model template files like scenes, articles, action responses, custom behaviors, and timelines in a particular order. We revised the user interface of the authoring wizard to include the model template validation in a particular order. The highlighted pane using a blue outline clearly illustrates the revision. Each stage requires successful validation to enter into a new specification stage, i.e., unless scene model template validation is unsuccessful, one cannot navigate to the article model template. Upon completing all stages, the requirement analysts can navigate all the stages to go through the individual specifications in detail. This revision helps requirement analysts review and specify requirements in stages and trace them to the initial specification.
- Project management: In our initial VReqST edition, the requirement specification was static and linear, i.e., a requirement analyst must specify and download the

finalized specifications. However, in the revised version, we included user-management and project-management modules to manage the specification group for a particular project. This will help requirement analysts save, maintain, and revise the specifications for individual projects. As part of this revision, we can now create a new project, share it with others, and edit/revise in the future based on the changes to our business needs.

- In-page validation messages: We revised our *ad hoc* message regarding validation status on demand on the same page rather than performing validation at the end of all stages. With the inclusion of multi-stage authoring, we implemented successful and failure messaging as in-page status messages.
- We have converted the underlying MongoDB instance from a private cloud cluster to a public cloud cluster so that the VR practitioners can download the underlying DB with the core installation instance and re-point it to their private cloud cluster instances. This will help the VR community practitioners to store their requirement specifications as private.
- Other minor revisions include font changes and text re-sizing across different pages, including additional fields on the registration page and a few minor implementation changes to make it simpler to deploy.

Iteration 2: The detailed changes conducted as part of our iteration 2 are listed below. We included all the required changes in the form of figures as part of Supplementary Material.

- Customizing validator files: Until the iteration 1 version of VReqST, the requirement analyst had to rely on the core model template validator for each validation stage, i.e., scenes, articles, action responses, custom behaviors, and timelines. The model template validator must be updated directly from the backend if they wish to customize. However, as part of this revision, we provisioned a customizing model template validator through UI. We also provide the ability to choose the custom validator while authoring the specification during project creation. Thus, as part of this iteration, the requirement analyst can upload a custom validator for each VR requirement specification stage and choose this custom validator while creating a requirement project. A new view includes "My projects" and "My validators," displaying a list of custom validators created by oneself, along with all model template validators developed by other requirement analysts using the same tool.
- Authoring states of articles: Until the previous iteration, the state for a given article is not included as part of the default article model template. As part of this iteration, the underlying article model template now supports the "state" of an article, where the requirement analyst will now have the ability to define the initial state, final state, and state transition of a given article based on the event performed with respect to a given timestamp.
- Other minor revisions in this iteration include adding text to the welcome page, providing documentation support as help, enabling the review of past and ongoing project details.

Iteration 3: The detailed changes conducted as part of our iteration 3 are listed below. We included all the required changes in the form of figures as part of Supplementary Material.

- Article and action/response template revisions: Until iteration 2, the requirement analysts must specify all the articles in a single article model template. As part of this iteration, we revised the article model template view stage by providing a new button to specify one article after another. As shown in figures from the Supplementary Material, the highlighted article stage in blue contains articles like *coin_1* and *coin_2*, listed beside the model template editor. Upon successfully specifying each article in step 2, the requirement analyst can click on the respective article to revise the specification before moving to the next stage, i.e., the action response stage. Step 3, highlighted in blue, i.e., action response, also has a similar update. Instead of authoring all action responses between two articles in a single action response model template file, we can now author them separately and view them individually. For example, under step 3, i.e., action response, we may find examples of specifications of action response between player and coin such as *player_grabcoin*, *player_steercart*, and *player_jumprailcart*.
- Behavior editor revisions: As part of this iteration, we revised the entire workflow of step 4, i.e., the *Behavior editor* formerly known as "*Custom JSON Step*." This step has three modes: 1) "*author behaviors*," where users can define authored behaviors and publish them using three sections, code constructs (if, if-else, for, while, etc.), a list of articles from the previous step, and a drag-and-drop interface for combining these elements; 2) "*view author behavior mode*" contains a list of behaviors from the "author behaviors" mode for review and management; and 3) "*publish*" mode contains a list of opens to drag and a list of behaviors to be included in the final validation list for the behavior step. These three modes can be visualized as part of the figures included in Supplementary Material.
- Other minor revisions include text re-sizing across different pages, adding additional fields to the validator file picker page, and implementing minor changes to simplify deployment.

After considering thorough feedback from the VR community, we updated the VReqST tool in three iterations and released it as an open-source tool. In contrast to this feedback, we surveyed the participants to understand the impact and overall experiences of using VReqST in practice. We present more details in this regard in the following subsection.

4.2 Observations—impact and experience survey

Between December 2023 and January 2024, we reached the 53 VR practitioners who participated in the VReqST iterative validation study to participate in an impact and experience survey to provide their overall experience of their team on using VReqST and their experiences in utilizing the specifications in practice. Of the 53 VR practitioners, 26 responded to the survey.

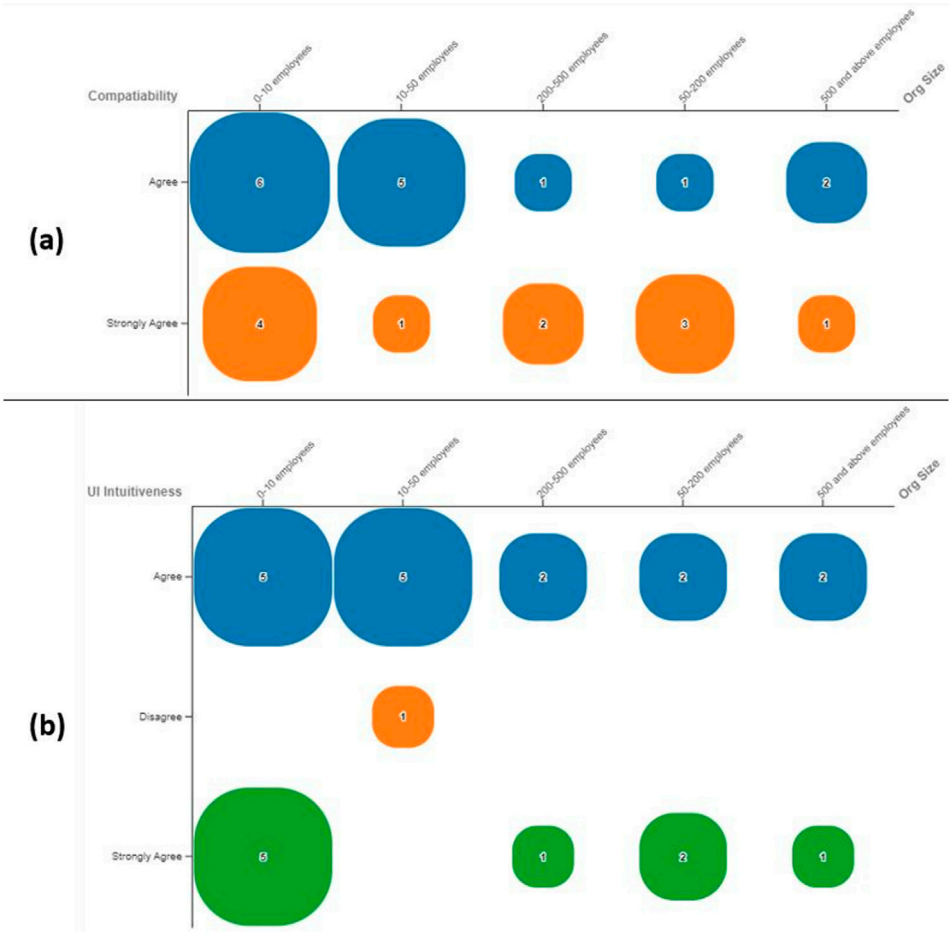

FIGURE 15
(A) Compatibility of VReqST with existing processes, grouped by organization size. **(B)** Ease of UI/UX of VReqST in practice, grouped by organization size.

The detailed observations from the survey are provided in the following sections.

4.2.1 Demography of participants

Most organizations that participated in this impact and experience survey have been building enterprise VR products for approximately 6 years on average, with 2 years being the minimum tenure of a given organization on a lower end and 20+ years for the maximum tenure of a given organization on a higher end on building VR products from countries US, UK, India, China, Europe, and Australia. Among the 26 participants, we have engineering manager(s), engineering directors with various capacities, product managers, technical architects, and software engineers who work in XR, an emerging technology domain. The participants of the survey primarily build VR products in the following domains: healthcare, defense, simulation, digital twins, interior design, inner environment design, industrial layout, corporate education, compliance training, entertainment, gaming, drug discovery, medical protocol design, gaming content, cinema, retail marketing, marketing campaigns, authoring tools, metaverse, fashion, ads, wearables, financial analytics, tourism, consulting, enterprise banking, and recreational tours. These practitioners are from Deloitte Digital, Plug XR, Cornerstone On Demand, SAP-XR, Samsung Studios, SkillSoft, UNITY Developer Group, ThoughtWorks Dev Group, Khronous Dev Community, and many other VR open source practitioners helped us provide their valuable feedback.

4.2.2 Compatibility

Figure 15A illustrates that most participants observe that the VReqST tool is compatible with their existing processes and can be easily integrated without any hassle. Primarily, small organizations with an organization size of less than 200 employees have found VReqST to be highly compatible with their regular VR product development processes.

4.2.3 Ease of use (UI/UX)

Figure 15B illustrates that most participants observe that the VReqST tool is intuitive, considering its user interface and overall user experience seem manageable in practice. Small organizations with an organization size of fewer than 200 employees have found VReqST to be highly consistent with its UI design and ease of use in practice.

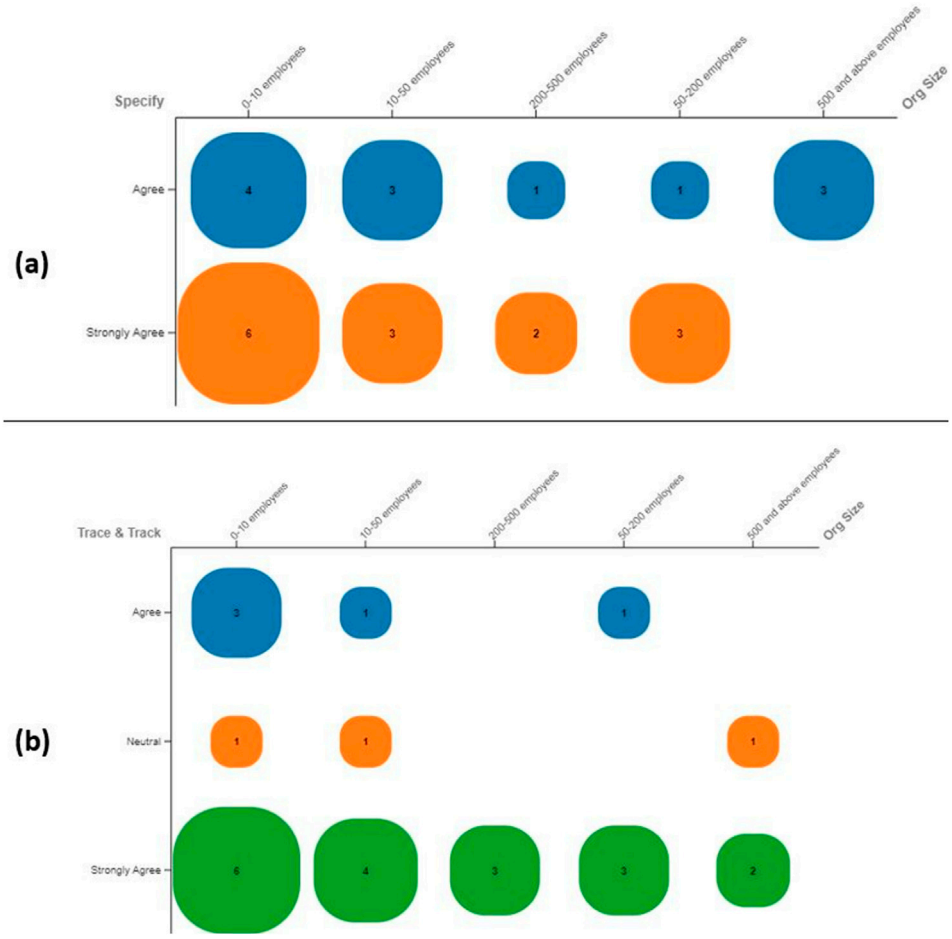

FIGURE 16
(A) Ability to specify VReqST specifications by frequency of usage across organizations. **(B)** Ability to track VReqST requirements by frequency of usage across different classes of organizations.

4.2.4 Frequency of use

Figure 16A illustrates that most participants who started using VReqST frequently or occasionally for authoring requirement specifications for their VR products mentioned that VReqST can specify requirements in detail, irrespective of the organization's size. In almost all cases, the VReqST is highly capable and easy to specify requirements, or it is manageable compared to their conventional approaches.

4.2.5 Feature tracking

Figure 16B illustrates that most participants who started using VReqST frequently or occasionally for authoring requirement specifications for their VR products mentioned that VReqST can easily track and trace requirements, irrespective of the organization's size. In almost all cases, the VReqST tool is highly capable and makes it easy to track and trace requirements to the overall features across the requirement specifications of VR products, especially when compared to conventional approaches.

4.2.6 Extendability

Figures 17A, B illustrate that most participants who started using VReqST for authoring requirement specifications for their VR products observed that VReqST is highly extendable and adaptable in practice. It can be extended to accommodate new model attributes as the underlying model validator can be customized for domain- or context-specific applications. Most small and mid-range organizations working on context-specific VR products observed VR as a better alternative to specifying requirements to meet their business goals.

4.2.7 Additional observations

Along with the survey questionnaire, we asked the survey participants to share their additional insights on adopting and using VReqST. The following subsection illustrates detailed responses in participants' words, which we have coded into specific themes for better illustration.

4.3 VReqST extensions in VR product development

With the VR community practitioners' exposure to VReqST, the following ideas are suggested as future extensions to the VReqST tool by the VR community to ease overall VR software product development. We detailed them as follows.

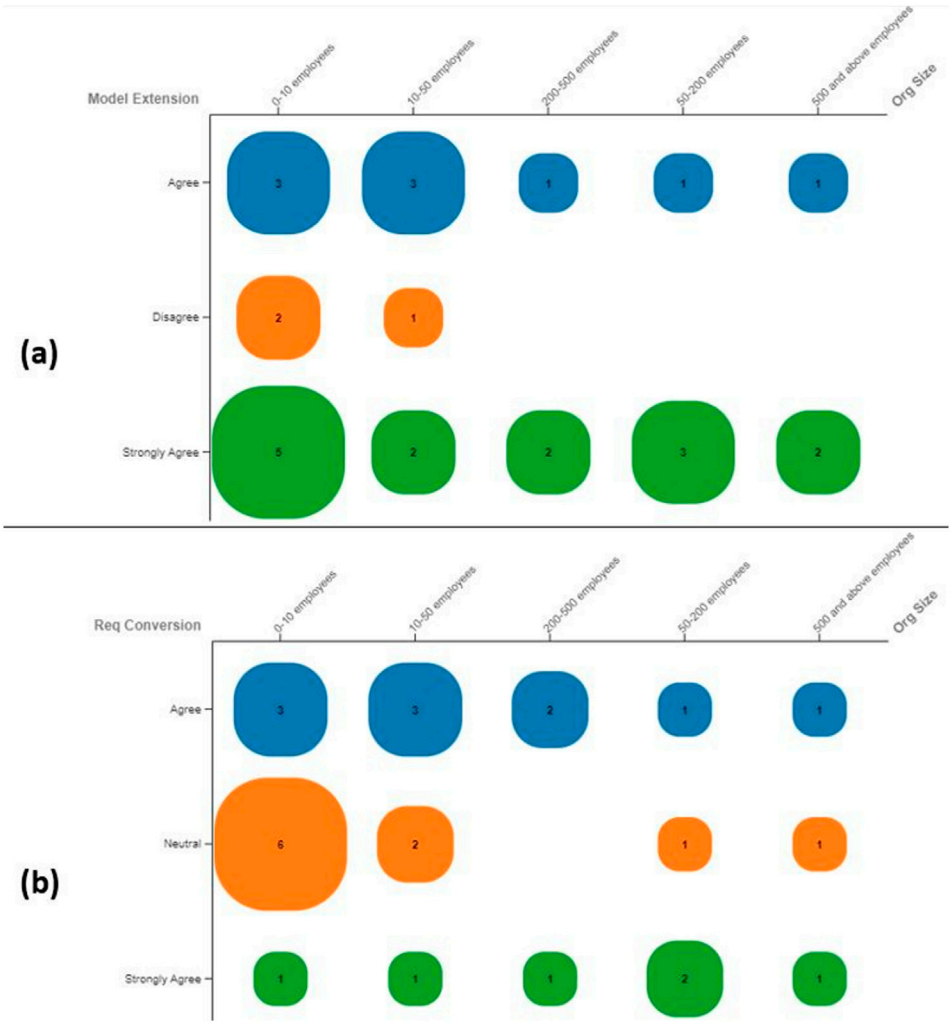

FIGURE 17
(A) Ability to extend VReqST to domain-specific applications, grouped by the ability to specify requirements based on conventional standards across different classes of organizations. **(B)** Ability to convert conventional specifications to VReqST specifications.

4.3.1 Code generation

VReqST-based requirement specifications can be extrapolated further on generating model-driven code generation tools for scenes and three-dimensional articles. Irrespective of the VR technology stack, VReqST specifications can be used to develop new tools that can generate alternate versions of code snippets for behaviors that could be generated on-demand during the scene play. It is one of the critical future aspects of extending VReqST into automated scene generation and behavior generation on run time.

- In most cases, VReqST specifications can be integrated into an automation pipeline. This is yet to be explored. We are considering expanding the underlying template to support security requirements. [**P4**]
- It helps consultants gather requirements. However, more automation is needed to channel the JSON specifications to code generation and testing. [**P12**]
- Very detailed. This tool will have no value unless it is integrated into an automation pipeline. There is need to build supporting tools like co-pilot to generate code with VReqST specifications. [**P15**]
- It provides the ability to record requires specifications with higher precision. Integration into an automated delivery pipeline is necessary for more adoption. [**P17**]
- We recently entered the XR market with a garment wearable product. We aim to integrate the JSON specifications into asset generation using AI models. Thanks for keeping this open source. [**P20**]

4.3.2 Test case generation

As VReqST curates the requirements in detail, automated test case generation of scene dynamics, article properties, action responses, behaviors, and timeline of events can be effortlessly explored to comprehend the quality of a delivered VR scene. A new, novel customizable software testing framework can be developed as an external wrapper around VReqST to generate on-demand test cases during run-time to improve the performance of the VR scene.

- We started as pilot project with two requirement analysts. We found VReqST to be high extendable. The generated specifications can be used as input source for our test case generation system to conduct unit and integration testing. We are planning to extend our specifications for automated design validations. Thank you for collaboration. [**P1**]
- We have been building games for approximately 20 years now using highly templatized terrain data and an internal asset repository. We might have to perform backward re-engineering of existing assets to align with this specification for automated scene generation, test case creation, and interaction-flow testing. This tool will definitely help newcomers. Good luck with the tool. [**P23**]

4.3.3 Risk and dependencies

With detailed VReqST requirements, the requirement analyst can determine the risks of prioritizing or extending new features. The requirement analyst can easily understand the overall dependencies of articles and their action responses on each other. These dependencies will help the requirement analyst estimate the consequences of the continuous evolution of a feature in VR and contribute toward a higher-quality VR product.

- Although it requires some effort, VReqST output is a good input source for our low-code platform. Most of the asset and behavior generation can be automated to some extent. The specifications are easy to maintain and version. Most of our designers rely on the JSON text to go through the specifications for revisions. [**P2**]
- It is easy to integrate into our established processes. Small requirements are easy to author and can later be traced to a large specification. It helps is maintaining a requirement repository for dedicated projects related to assets and terrain definitions. It also helps re-draw scope and assess risks associated with failing to meet the requirement. [**P9**]
- It is plug-and-play with internal VR authoring tools and makes it easy to determine the risk of requirement failure during implementation. However, it is difficult to read requirements in the JSON format. A better way to read the VReqST specification in detail is needed. [**P10**]
- VReqST may not be very useful as we have curated fixed, pre-defined visualization templates. However, it helps when we have to redesign a new visualization template. The reusability and traceability of reporting features are easy to observe from the specification to the template. It may require less adaption but could become vital in specific business cases. We suggest extending an AI mode to predict/propose visualization templates for recorded VReqST specifications. [**P18**]

4.3.4 Traceability

As the VReqST requirement specifications establish a relationship between VR concepts like scenes, articles, action responses, and timelines, the changes made to one concept will reflect across other concepts. These specifications support design traceability, helping understand the rationale behind design decisions. New tools can be developed alongside VReqST to handle code traceability, aiding the understanding of code segments and assists with debugging. Such new tools can support test traceability to help testers identify gaps in test-case coverage.

- More effort is needed to define our product-specific validator. We cannot use the tool with our own business/context-specific validator. The specifications are easy to version and trace. For RAs, it is difficult to read JSON specifications as they are not technically skilled. [**P5**]
- Traceability of feature is the key. It is easy to author micro requirements. [**P24**]

4.3.5 Maintenance and portability

With substantial requirements, the challenges of VR platform portability, i.e., porting VR scenes developed using a particular technology stack to another, can be mitigated to some extent. As VReqST can distinguish platform-independent requirements from platform-dependent ones, VR developers can now maintain platform-independent source code separately from the regular code branch. This practice may improve the maintainability and versioning of VR source code.

- We are still in early stages of adoption. Our analysts are not tech savvy to use this tool. However, our developers like it as it helps them understand requirements with higher accuracy. We still need to explore its capabilities for portability. [**P3**]
- When we started, we had to author our own validator. It is easy to integrate but requires a lot of early effort. The initial steps are challenging. A strong community support system is required to reuse and share artifact specifications. There is a need for an asset specification repository like GitHub. [**P6**]
- We mostly use it for AR advertisement campaigns. It is easy to use and manage specifications. [**P25**]
- We can also author requirements for AR applications by updating the underlying validator. The specifications are portable between VR-only and MR applications, even if they were originally authored for AR applications. However, extra effort is needed to develop an underlying template for our own business need. [**P8**]
- It requires a lot of effort to understand and update the domain-specific validator. It requires more community support. [**P11**]
- The requirements are too detailed. It is difficult to illustrate large terrains with such high specificity. However, it is easy to author micro-requirements. [**P14**] Our use-cases are highly domain-specific. We need to put a lot of efforts into developing the underlying validator. For now, we are better with our conventional approach to capture requirements in templates. [**P22**]

4.3.6 New types of requirements

The current VReqST version does not facilitate security, privacy, and usability requirements. The VR community recommends facilitating additional features to VReqST, including customizable security, privacy, and usability requirements, as part of the new model template. These requirement types are non-generic and domain-centered. They may vary from application to application. As VReqST is customizable, security, privacy, and usability requirements can be included beyond the bare-minimum model

template. It can be managed by domain-specific requirement analysts to accommodate such new requirement types by composing a custom model template on par with the bare-minimum model template.

- The specificity of requirements is maintained through this tool. However, it requires training as our BAs are not technical enough to author specifications in JSON. We request a form-based template that can convert to JSON in backend. [**P7**]
- We request the inclusion of a privacy-related specification validator by default. This would help cover compliance use-cases easily. [**P13**]
- Fashion requires clear specifications, and VReqST fills that gap. While it is slightly painful to write our custom validator, it is easy to maintain and re-trace the specifications. Strong collaboration is needed among XR community partners to share open-source asset specifications for better reusability. [**P16**]
- Currently, our plain English requirement specifications are being converted to VReqST specifications. It helps us create large terrain templates for future touring projects. The behavior editor is the coolest feature. It is easy to author and track linkages between the interactions. [**P19**]

4.4 Proof of concept: specifying requirements for depression detection application using VR

4.4.1 Background

Between January 2023 and October 2023, researchers from the Software Engineering Research Center and Cognitive Science Laboratory at IIIT Hyderabad worked toward developing multiple virtual reality scenes for the detection or intervention of depression using VR technology. These VR scenes are used as a medium to evoke emotions that are deemed to result in positive, negative, and neutral emotions. These three emotions are evoked using positive, negative, and neutral environments. Figure 18 illustrates the positive scene, negative scene, and neutral scene with top view and side views. The requirement specifications of these VR scenes are first authored using VReqST; they are revised and then finalized for design and development. These VR scenes are developed in UNITY Game Engine (Unreal Game Engine, 2022) and are visualized using HTC Vive Pro 2. This headset has a dual RGB low persistence LCD screen with a resolution of 2,448 × 2,448 pixels per eye (4,896 × 2,448 pixels combined) and a refresh rate of 90/120 Hz. It has a field of view of 120° and requires two compatible base stations and two compatible motion controllers. We used a PC with 16 GB of RAM and a 970 graphics card.

4.4.2 Experiment

The three developed VR scenes are designed to evoke the participants' positive, negative, and neutral emotions while locomoting in the virtual environment. While the participant is under locomotion under the influence of a VR scene, the participant's gait pattern is captured using a VelGmat (Wani et al., 2022), i.e., a Velostat-based gait mat that captures gait pressure. The gait pattern under the influence of positive, negative, and neutral scenes will reveal the levels of depression in a given participant as the scene evokes emotions. The participant is required to walk around in the environment once clockwise around the right shelf and once anti-clockwise around the left shelf, and in each round, while returning, he/she is required to pick up the specified object and place it into the respective crate.

4.4.3 VR scene requirements

All three environments have a 10 × 2 foot aisle with six-foot-high shelves on both sides, two feet apart. There are two similar aisles on the right and left that allow the players to circle the right and left shelves clockwise and anticlockwise, respectively. The walking area is separated from the respective terrain by a glass wall. The right aisle has an opaque wall with images of positive and negative environments. Both shelves are stacked with four objects on three different levels. The aisles on the sides have crates in the corners.

- Positive environment: Figure 18 illustrates the top and side views of the positive scene. The goal of the positive scene is to evoke happiness in the participant. It is primarily achieved using bright and warm lighting sources with saturated colors, accompanied by pleasant background music. The environment is filled with forest terrain with trees around a central plan with two racks separated by a walkable area. The racks contain few objects. The two large crates on two ends of the center pane are on one side of the scene, whereas the other end of the center pane contains images floating on a virtual wall. The objects in the scene are primarily under color-grade shades of light blue and light pink for better differentiation. The images on the display wall are selected from the International Affective Picture System (IAPS) database with high valance ratings (above 50). IAPS is a database of pictures designed to provide a standardized set of pictures for studying emotion and attention. It is widely used in psychological research. The tree assets used in the forest were imported from the Unity Demo URP terrain.
- Negative environment: Figure 18 illustrates the top and side views of the negative scene. The goal of the negative scene is to evoke sadness in the participant. It is primarily achieved using dark and low-intensity lighting sources with dull colors, along with disturbed background music. The environment is filled with forest terrain with trees around a central plan with two racks separated by a walkable area. The two large crates on two ends of the center pane are on one side of the scene. The racks are bulkier and contain too many objects, thus making the space feel more congested. One end of the center pane contains images floating on a virtual wall. The objects in the scene are primarily under color-grade shades of dark brown and dark green, making them difficult to differentiate. The images on the display wall depict decay and suffering, selected from the IAPS database with very low valance ratings (less than 50). IAPS is a database of pictures designed to provide a standardized set of pictures for studying emotion and attention. It is widely used in psychological research. The tree assets used in the forest were imported from the Unity Demo URP terrain.

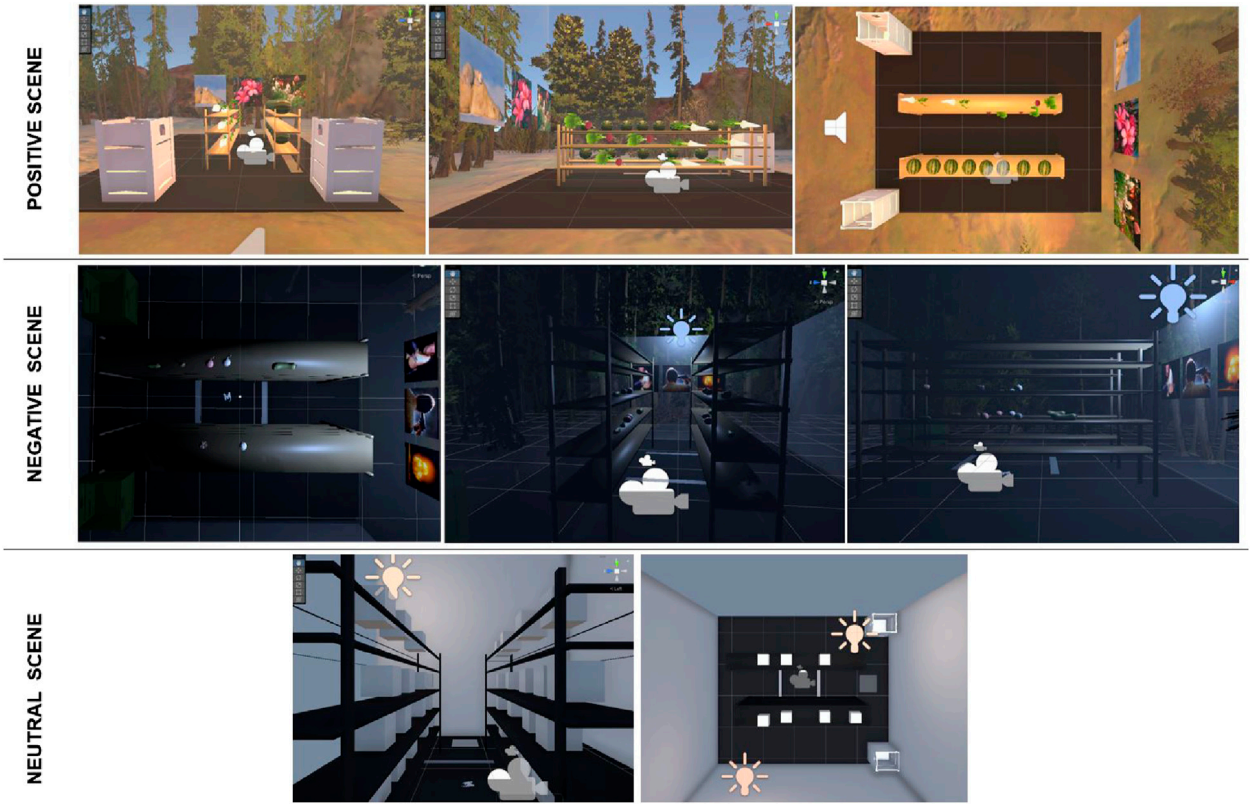

FIGURE 18
Scene flow of positive, negative, and neutral scenes in mental health VR applications.

- Neutral environment: Figure 18 illustrates the top and side views of the neutral scene. This controlled environment is designed to collect participants' gait data without emotional elicitation in the VR. The scene contains black and white colors on all the objects, with no images on the wall or music.

4.5 Future work

Model-driven development (MDD) is a software engineering methodology that emphasizes the creation and use of domain models. These are conceptual models representing various aspects of a software system. They serve as blueprints for software development, guiding the process from design to code generation. MDD involves abstracting technical aspects such as logic, data models, and user interfaces into visual representations that can be easily manipulated. The ultimate goal of MDD is to enhance productivity, improve software quality, and facilitate collaboration among developers of varying skill levels. In the context of virtual reality software products, our role-based model template can be useful for formulating a model-driven development approach using domain-specific large language models and generative AI. Our role-based model template for comprehending the virtual reality technology domain is a backbone for formulating model-driven development for developing virtual reality software products. Using our role-based model template for VR, we developed an open-source requirement specification tool called VReqST. This tool can apprehend bare-minimum concepts related to the VR technology domain. It can generate precise and clear specifications that can become an input to our MDD pipeline for virtual reality. Figure 19 describes a development pipeline for virtual reality software. During the requirement phase, VReqST can be used to specify requirements, which are then used as input to a three-dimensional large learning model (LLM) that is specific to the VR domain. The input will be in a JSON format. In the design phase, the 3-D LLM is trained using 3-D object data sources that offer data in the form of point clouds, voxels, or meshes to generate multiple desired articles, mock design templates for terrain, three-dimensional environment, and respective action response flow.

The output of the 3-D LLM can be used as input to a low-code platform that generates the behavior script for specific VR game engines and creates a consolidated behavior script library for future reference. A model-based testing protocol can be formulated for these behavior scripts based on the generated code snippets. The overall 3-D run-through can be evaluated using a machine learning BOT through point-of-view testing. The tested artifact can now be deployed using a model-based deployment approach, where changes reflected in the model (scene artifacts, article artifacts, action response artifacts, behavior artifacts, and timeline artifacts) can be deployed asynchronously in an incremental approach to production. However, despite various automated processes involved in the proposed overall MDD pipeline for virtual reality software development, it is partially a human-in-a-loop process. Human validation is required in a few stages to validate the

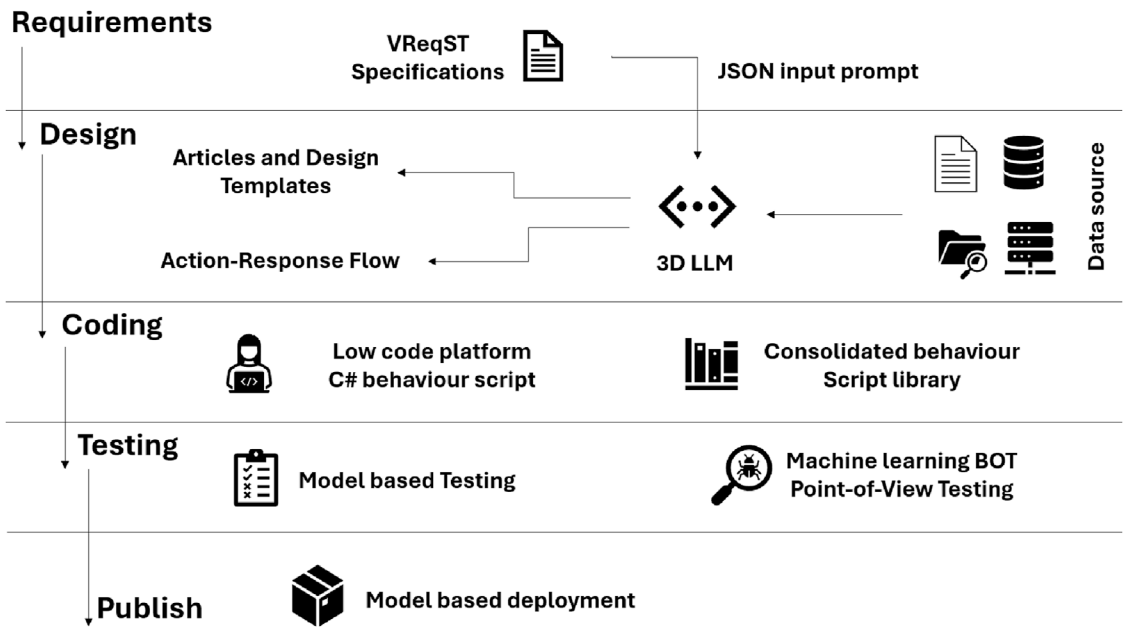

FIGURE 19
Model-driven development using LLM and GenAI for virtual reality software products.

TABLE 4 Comparison of VReqST with existing VR specification tools.

VR specification tool	STATEMENT	VR-WISE	Common scene definition	CLEVR	Mental model	VReqST
Tool type	Message sequence diagram	Conceptual model	Fixed scene definition	VR system conceptual model	VR system conceptual model	VR domain meta-model
Can specify requirements	Yes	Yes	Yes	Yes	Yes	Yes
Support requirement traceability	No	No	No action	No	No	Yes
Reusability of requirements	No	No	No	No	No	Yes
Ease of usage of the tool	Easy	Moderate	Easy	No study available	No study available	Easy
Supports domain-specific/context-specific requirements	No support	Supports	No support	Supports	Customizable	Full Support
Requirement versioning	No support	Partially	No support	Partially	No support	Full support
Portable to new VR engines	Low	Medium	Low	Low	Low	High
Maintenance	High	Medium	High	High	Medium	Low
Current fate	Obsolete	Obsolete	Needs revision	Needs revision	Needs revision	Available and revisable

exactness and quality of the resultant VR scene. This hypothesis requires a lot of conviction to execute at a large scale.

5 Related work

Although VR applications have much attention in practice, as a domain, they still require wider adoption. Platform fragmentation and hardware dependencies are causing serious adoption issues. Early studies in requirement specification in VR are either human-centered or process-oriented. The VR community widely practices functional software requirement specification documentation, unified modeling language/legacy software requirement specification documentation (Ragkhitwetsagul et al., 2022), templatized specifications (Okere et al., 2016), and task tree documentation (Colombo et al., 2020). These approaches do not facilitate detailed specifications that are VR-centered, and it depends on the requirement analyst's ability to articulate the requirements effectively. Thus, such approaches are subjective and arbitrary in practice. Other approaches like common scene definition (Belfore

et al., 2005), conceptual VR prototyping (Manninen, 2002), and mental modal technique (Mohd Muhaiyuddin and Awang Rambli, 2014) are distinct requirement specification methods that are used on sample VR scenes and studying spatial presence. They are specific to simulation-based VR applications and are not extendable to other VR applications. Kim et al. (1998) were the first to explore the possibilities of creating a process-oriented approach for requirement engineering in VR. Early tools like STATEMENT (Kim et al., 1998) based on POSTECH and message sequence diagrams (MSDs) paved the way for VR requirement engineering automation. Pellens et al. (2005) proposed VR-WISE for modeling the static part of VR, from conceptual specification to code generation, Seo and Kim (2002) proposed a CLEVR model that considers the functions, behavior, hierarchical modeling, user task and interaction modeling, and composition reuse in early VR systems. Alinne et al. proposed a scene-graph-based approach for requirement specification and testing automation of VR products (Souza et al., 2018). Our previous work about an extensive systematic literature review on requirement engineering practices of VR software products (Karre et al., 2024) also illustrates that almost all the requirement specification approaches are either template-based or manual in practice. Most of these tools and approaches are obsolete and require a generalized revision to align with the contemporary state-of-the-art in VR as a technology domain. Amidst such challenges, we ideated and developed VReqST, a model-based tool to specify requirements for VR products to address the observed gaps for the VR community to adopt and practice. Table 4 illustrates the unique contributions of VReqST in comparison to existing VR requirement specification tools.

Data availability statement

The datasets presented in this study can be found in online repositories. The names of the repository/repositories and accession number(s) can be found in the article/Supplementary Material.

Ethics statement

Written informed consent from participants was not required to participate in this study in accordance with the national legislation and the institutional requirements.

Author contributions

SK: conceptualization, data curation, formal analysis, funding acquisition, investigation, methodology, project administration, resources, software, supervision, validation, visualization, writing–original draft, and writing–review and editing. YR: conceptualization, data curation, formal analysis, funding acquisition, investigation, methodology, project administration, resources, software, supervision, validation, visualization, writing–review and editing, and writing–original draft.

Funding

The author(s) declare that financial support was received for the research, authorship, and/or publication of this article. This effort was carried out at the Design Innovation Center, IIIT Hyderabad, India, partially funded by the Ministry of Education, Government of India.

Acknowledgments

The authors thank the VR practitioners from SAP-XR, Deloitte Digital Laboratories, ThoughtWorks, Samsung Studios, UNITY Dev Group, and Khronos Dev Community for participating in the empirical study to formulate and sharing their insights.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Generative AI statement

The author(s) declare that Generative AI was used in the creation of this manuscript. We acknowledge that image as part of Figures 11–iv image generated by OpenAI's DALL-E 2 for reference purpose only, December 15, 2023.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors, and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

Supplementary material

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/frvir.2024.1471579/full#supplementary-material

References

AframeJS (2022). AframeJS documentation.

Amazon Lumberyard (2022). *Lumberyard documentation*. Amazon Web Services.

Anwar, M. S., Choi, A., Ahmad, S., Aurangzeb, K., Laghari, A. A., Gadekallu, T. R., et al. (2024). A moving metaverse: Qoe challenges and standards requirements for

immersive media consumption in autonomous vehicles. *Appl. Soft Comput.* 159, 111577. doi:10.1016/j.asoc.2024.111577

Arshad, H., Shaheen, S., Khan, J., Anwar, M., Khursheed, K., and Alhussein, M. (2023). A novel hybrid requirement's prioritization approach based on critical software project factors. *Cognition Technol. Work* 25, 305–324. doi:10.1007/s10111-023-00729-3

Belfore, L. A., Krishnan, P. V., and Baydogan, E. (2005). "Common scene definition framework for constructing virtual worlds," in Proceedings of the 37th Conference on Winter Simulation (Winter Simulation Conference), WSC'05, Orlando, FL, December 4, 2005, 1985–1992. doi:10.1109/wsc.2005.1574477*Proc. Winter Simul. Conf. 2005.*

Brennesholtz, M. (2017). VR/AR standards – are we confused yet?

Colombo, C., Blas, N. D., Gkolias, I., Lanzi, P. L., Loiacono, D., and Stella, E. (2020). An educational experience to raise awareness about space debris. *IEEE Access* 8, 85162–85178. doi:10.1109/ACCESS.2020.2992327

CryEngine Cry (2021). *CRYENGINE programming documentation*. Crytek Technologies.

Filho, O. V. S., and Kochan, K. G. (2001). "The importance of requirements engineering for software quality," in Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics: Information Systems Development-Volume I - Volume I (IIIS) (ISAS-SCI '01), Orlando, FL, July 22–25, 2001, 529–532.

Gervasi, V., and Nuseibeh, B. (2000). "Lightweight validation of natural language requirements: a case study," in Proceedings Fourth International Conference on Requirements Engineering, ICRE 2000 (Schaumburg, IL: Cat. No.98TB100219), 140–148. doi:10.1109/ICRE.2000.855601

Gonzalez-Perez, C., and Henderson-Sellers, B. (2008). *Metamodelling for software engineering*. Wiley Publishing.

Google Inc. (2010). COLLADA - digital asset and FX exchange schema.

[Dataset] Gramlich, T. (2022). The future of work and virtual reality.

Group, K. (2019). The OpenXR specification 1.0.24.

Hoda, R. (2021). Socio-technical grounded theory for software engineering. *IEEE Trans. Softw. Eng.* 48, 3808–3832. doi:10.1109/TSE.2021.3106280

[Dataset] Kandhari, K. (2023). Vr bowling alley game.

[Dataset] Karre, S. A. (2024a). Vreqst - online documentation for virtual reality requirement analysts.

Karre, S. A. (2024b). VReqST sample specifications.

Karre, S. A., Mittal, R., and Reddy, R. (2023). "Requirements elicitation for virtual reality products - a mapping study," in Proceedings of the 16th Innovations in Software Engineering Conference ISEC'23, Allahabad India, February 23–25, 2023 (New York, NY, USA: Association for Computing Machinery). doi:10.1145/3578527.3578536

Karre, S. A., Neeraj, M., and Reddy, Y. R. (2019). "Is virtual reality product development different? an empirical study on vr product development practices," in Proceedings of the 12th Innovations on Software Engineering Conference (ISEC), Pune, India, February 14–16, 2019 (New York, NY: ACM). doi:10.1145/3299771.3299772

Karre, S. A., Pareek, V., Mittal, R., and Reddy, Y. R. (2022). "A role based model template for specifying virtual reality software," in In proceedings International Workshop on Virtual and Augmented Reality Software Engineering, in conjucture with Automated Software Engineering (ASE 2022), Oakland Center, MI, October 10–14, 2022 (New York, NY: : ACM). doi:10.1145/3551349.3560514

Karre, S. A., Reddy, Y. R., and Mittal, R. (2024). Re methods for virtual reality software product development: a mapping study. *ACM Trans. Softw. Eng. Methodol.* 33, 1–31. doi:10.1145/3649595

Kim, D., France, R. B., Ghosh, S., and Song, E. (2003). "A role-based metamodeling approach to specifying design patterns," in 27th International Computer Software and Applications Conference (COMPSAC 2003): Design and Assessment of Trustworthy Software-Based Systems, Dallas, TX, 3-6 November 2003 (IEEE Computer Society), 452. doi:10.1109/CMPSAC.2003.1245379

Kim, G. J., Kang, K. C., Kim, H., and Lee, J. (1998). "Software engineering of virtual worlds," in Proceedings of the ACM Symposium on Virtual Reality Software and Technology VRST '98, Taipei, Taiwan, November 2–5, 1998 (New York, NY, USA: Association for Computing Machinery), 131–138. doi:10.1145/293701.293718

LaValle, S. M. (2020). *Virtual reality*. Cambridge University Press.

Levy, J. R., and Bjelland, H. (1994). *Create your own virtual reality system*. United States: McGraw-Hill, Inc.

Manninen, T. (2002). Contextual virtual interaction as part of ubiquitous game design and development. *Personal. Ubiquitous Comput.* 6, 390–406. doi:10.1007/s007790200044

Martin, D., Malpica, S., Gutierrez, D., Masia, B., and Serrano, A. (2022). Multimodality in vr: a survey. *ACM Comput. Surv.* 54, 1–36. doi:10.1145/3508361

Mittal, R., Karre, S. A., Gururaj, Y. P. K., and Reddy, Y. R. (2022). "Enhancing configurable limitless paths in virtual reality environments," in 15th Innovations in Software Engineering Conference ISEC 2022, Gandhinagar India, February 24–26, 2022 (New York, NY, USA: Association for Computing Machinery). doi:10.1145/3511430.3511452

Mohd Muhaiyuddin, N. D., and Awang Rambli, D. R. (2014). "Navigation in image-based virtual reality as the factor to elicit spatial presence experience," in 2014 International Symposium on Technology Management and Emerging Technologies, Langkawi Island, Malaysia, May 27–29, 2014, 349–354. doi:10.1109/ISTMET.2014.6936532

Nebeling, M., and Speicher, M. (2018). "The trouble with augmented reality/virtual reality authoring tools," in 2018 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct), Munich, Germany, October 16–20, 2018, 333–337. doi:10.1109/ISMAR-Adjunct.2018.00098

Okere, H. C., Sulaiman, S., Rambli, D. R. A., and Foong, O.-M. (2016). A multimodal interaction design guidelines for vr foot reflexology therapy application. *Int. J. Oper. Res. Inf. Syst.* 7, 74–91. doi:10.4018/IJORIS.2016070105

Pellens, B., Bille, W., De Troyer, O., and Kleinermann, F. (2005). "Vr-wise: a conceptual modeling approach for virtual environments," in Proceedings of the methods and Tools for Virtual Reality Workshop (MeTo-VR), 1–10.

Ragkhitwetsagul, C., Choetkiertikul, M., Hoonlor, A., and Prachyabrued, M. (2022). "Virtual reality for software engineering presentations," in *2022 29th asia-pacific software engineering conference (APSEC)*, 507–516. doi:10.1109/APSEC57359.2022.00072

Seo, J., and Kim, G. J. (2002). "Design for presence: a structured approach to virtual reality system design," in *Teleoperators virtual environ*, 378–403.

Sherman, W. R., and Craig, A. B. (2003). "Introduction to virtual reality systems," in *Understanding virtual reality. The morgan kaufmann series in computer graphics*. Editors W. R. Sherman and A. B. Craig (San Francisco: Morgan Kaufmann), 70–73. doi:10.1016/B978-155860353-0/50003-3

Sony Computer Entertainment Inc. (2004). COLLADA - digital Asset and FX exchange schema. Khronos Group.

Souza, A., Nunes, F., and Delamaro, M. (2018). An automated functional testing approach for virtual reality applications. *Softw. Test. Verification Reliab.* 28. doi:10.1002/stvr.1690

Strauss, A., and Corbin, J. (1967). Discovery of grounded theory.

UNITY3D (2022). *Unity3D manual - offline documentation*. UNITY Technologies.

Unreal Game Engine (2022). *UnRealEngine 5 documentation*. New York, NY: EPIC Games Inc.

W3C (1997). VRML virtual reality Modeling Language. Web3D consortium.

Wani, M. W., Gururaj, Y. P., P, V., Karre, S. A., Reddy, R., and Azeemuddin, S. (2022). "Velgmat: low cost gait mat for stance phase calculation," in 2022 IEEE Sensors, 1–4. doi:10.1109/SENSORS52175.2022.9967332

X3D (1997). Extensible 3D. Web3D consortium.