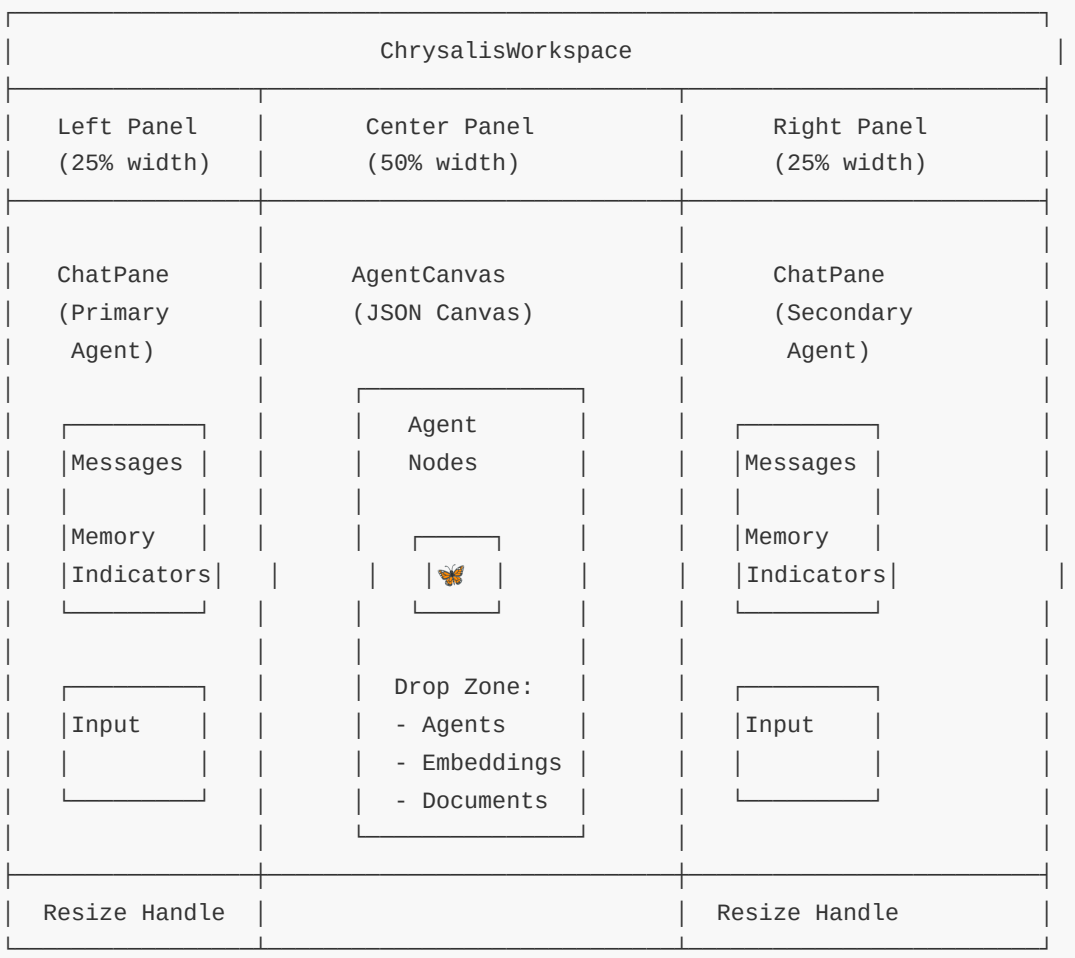
Chrysalis UI + MemU Integration Plan

Executive Summary

This document describes the three-frame UI architecture for Chrysalis with MemU memory integration, enabling agents to learn from builder pipelines and chat about their knowledge.

Architecture Overview



Components Implemented

1. Three-Frame Layout (`ChrysalisWorkspace.tsx`)

- **Left Panel**: ChatPane bound to Primary Agent
- **Center Panel**: AgentCanvas for visual agent management
- **Right Panel**: ChatPane bound to Secondary Agent (optional)
- **Resizable Panels**: Drag handles between panels

2. MemU Memory Layer (`MemUAdapter.ts`)

Four-tier memory architecture:

- **Working Memory**: Short-term session context (TTL-based)
- **Episodic Memory**: Past experiences and events
- **Semantic Memory**: Facts and knowledge
- **Procedural Memory**: Skills and learned procedures

3. Legend Embedding Loader (`LegendEmbeddingLoader.ts`)

Bridges builder pipeline outputs to MemU:

- Loads `*_embeddings.json` files from Builder pipelines
- Converts knowledge embeddings to semantic memories
- Converts skill embeddings to procedural memories
- Tracks loaded legends for learning visibility

4. Agent Chat Controller (`AgentChatController.ts`)

Connects agents to chat panes with memory:

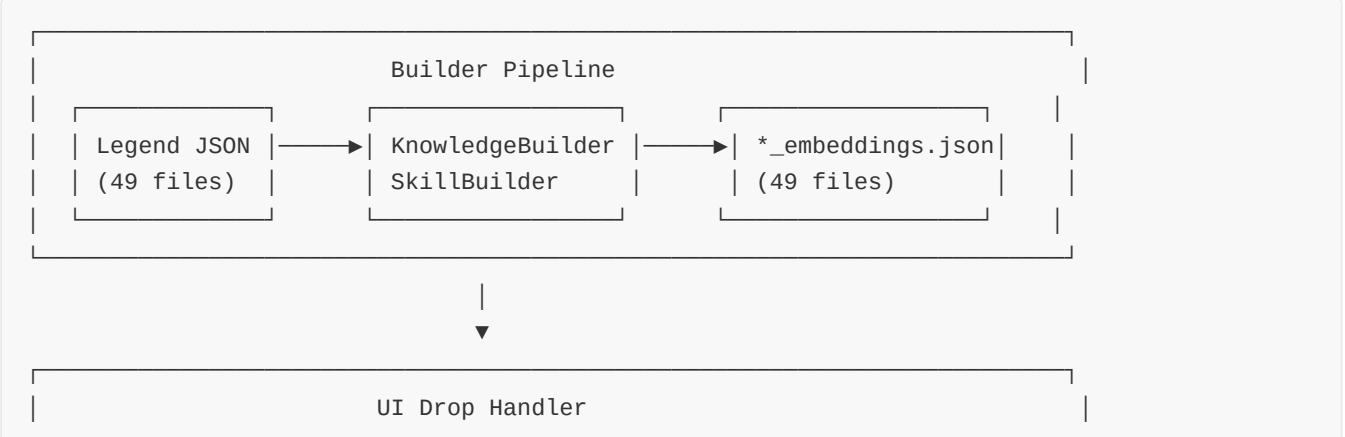
- Memory recall during conversations
- Learning from interactions
- Learning status reporting methods
- Methods to discuss learned knowledge

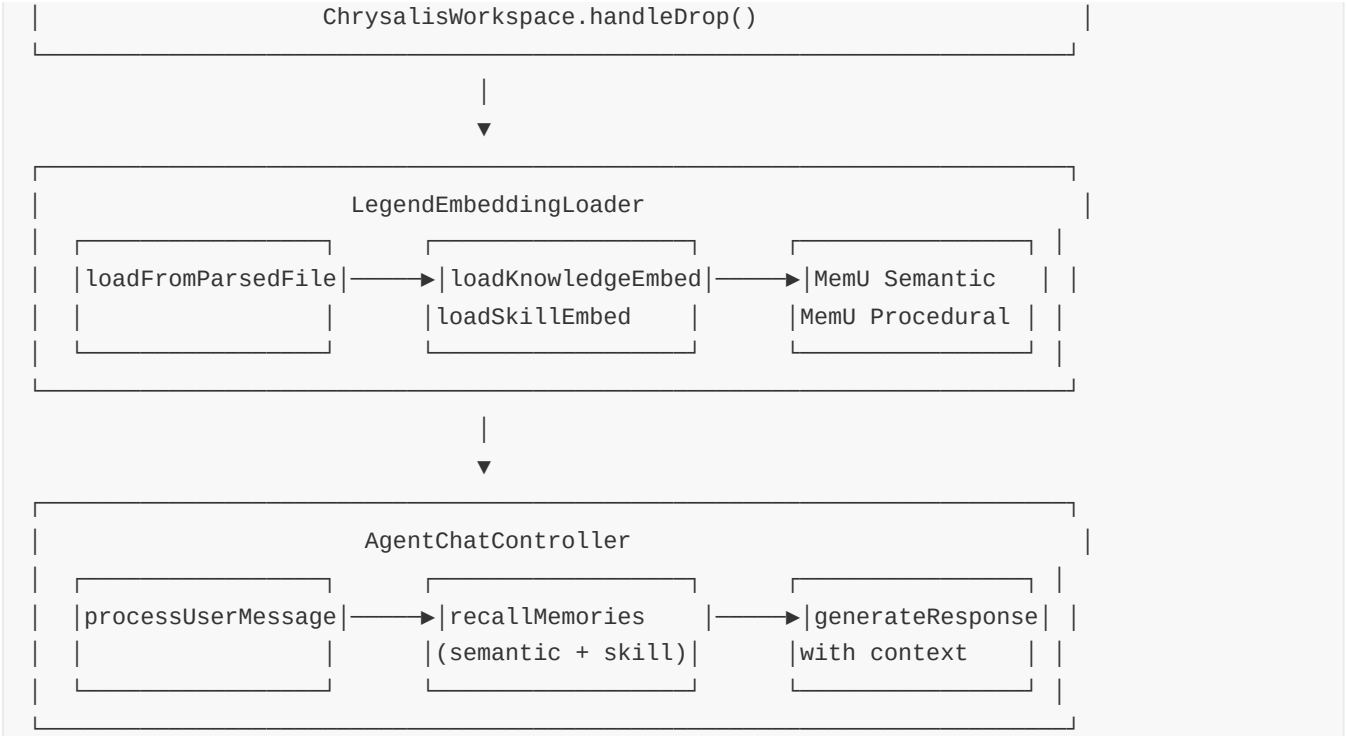
5. Agent Learning Pipeline (`AgentLearningPipeline.ts`)

Orchestrates learning activities:

- Conversation → Episodic → Semantic extraction
- Document processing for knowledge acquisition
- Skill learning from action patterns

Data Flow





API Reference

LegendEmbeddingLoader

```
const loader = new LegendEmbeddingLoader(memUAdapter, agentId);

// Load embeddings from parsed file
const result = await loader.loadFromParsedFile(embeddingData, options);

// Generate learning report
const report = loader.generateLearningReport();

// Search legend knowledge
const facts = await loader.searchLegendKnowledge('Ada Lovelace', 'algorithm');

// Get legend skills
const skills = await loader.getLegendSkills('Ada Lovelace');
```

AgentChatController (Learning Methods)

```
const controller = new AgentChatController(agent, 'left', memUAdapter, ...);

// Load legend embeddings
await controller.loadSingleLegend(embeddingData);

// Get learning report
const report = controller.getLearningReport();

// Check learned legends
const learned = controller.hasLearnedLegend('Ada Lovelace');

// Generate learning discussion
const response = await controller.generateLearningDiscussion('What do you know about
algorithms?');
```

File Structure

```
src/
├── components/
│   ├── ChrysalisWorkspace/
│   │   ├── ChrysalisWorkspace.tsx    # Main three-frame layout
│   │   ├── ChatPane.tsx              # Chat pane component
│   │   ├── types.ts                  # Type definitions
│   │   └── index.ts                  # Exports
│   └── AgentCanvas/
│       ├── AgentCanvas.tsx           # Center canvas component
│       ├── AgentNodeWidget.tsx       # Agent node rendering
│       └── AgentImportMenu.tsx       # Import UI
├── memory/
│   ├── MemUAdapter.ts                # 4-tier memory system
│   └── types.ts                      # Memory types
├── learning/
│   ├── AgentLearningPipeline.ts      # Learning orchestration
│   ├── ConversationMemoryManager.ts  # Conversation handling
│   ├── LegendEmbeddingLoader.ts      # Builder→MemU bridge [NEW]
│   └── index.ts                      # Exports
└── agents/
    └── AgentChatController.ts        # Chat + memory controller
```

Usage Example

```
import { ChrysalisWorkspace } from './components/ChrysalisWorkspace';

function App() {
  return (
    <ChrysalisWorkspace
      userId="user-123"
      userName="Developer"
```

```
      primaryAgent={{
        agentId: 'ada-lovelace',
        agentName: 'Ada Lovelace',
        agentType: 'primary',
      }}
      secondaryAgent={{
        agentId: 'ludwig-wittgenstein',
        agentName: 'Ludwig Wittgenstein',
        agentType: 'secondary',
      }}
      onMemoryEvent={(event) => {
        console.log('Memory event:', event);
      }}
    />
  );
}
```

Testing Steps

1. **Start the UI**: `npm run dev`
2. **Drop embedding file**: Drag `ada_lovelace_embeddings.json` to center canvas
3. **Verify loading**: Check console for "Loaded legend embeddings"
4. **Chat with agent**: Ask "What have you learned?"
5. **Verify recall**: Agent should reference loaded knowledge

Next Steps

1. [ ] End-to-end testing with actual embedding files
2. [ ] Add learning progress indicator in UI
3. [ ] Implement batch loading of all 49 legends
4. [ ] Add memory visualization in AgentCanvas
5. [ ] Connect to real LLM for response generation

Related Documents

- `memory_system/README.md` - Memory system v3.3.0
- `docs/FRONTEND_THREE_FRAME_FIVE_CANVAS_SPEC.md` - Original spec
- `scripts/process_legends.py` - Builder pipeline script