Lessons learned from using a networked entity system in game development

Abstract

In this work we analyze the usage of a previously presented networked entity system for developing multiplayer games. The source codes of the example cases are studied to observe the entity system usage in practice. The main goal is to evaluate the entity system on a conceptual level: does it succeed to provide a productive API for networked game development? What enhancements and supporting functionality could further aid application programming? (XXX NOTE: put the answers / conclusions to here when they exist, questions (also) to introduction?)

The entity system is implemented in the open source Tundra SDK which is used in the evaluation. Observations from game codes are divided into conceptual level entity system remarks, and issues related to the concrete particular implementation in Tundra currently.

Introduction

Networked application programming is generally more complex than standalone software development. The developer typically needs to deal with additional concepts, such as connections, and error handling and even conflicting situations in the distributed execution.

"Information hiding is a cornerstone principle of modern software engineering. Interfaces, or APIs, are central to realizing the benefits of information hiding, but despite their widespread use, designing good interfaces is not a trivial activity. Particular design choices can have a significant detrimental effect on quality or development productivity." cmu-api_failures

An entity system for networked application development has been put forth in [Alatalo2011]. This article draws lessons learned from how that system has been put to practice in a few different game development projects. The analysis is done by studying the source codes of the games, and by describing issues encountered in the development.

How can a conceptual design of an entity system be really evaluated? How can we know how well a platform supports actual networked game development? These are not easy questions, but the answers would really help us concretely in game and platform development. We do not claim to provide final answers here. The area of software and API complexity analysis has however made interesting progress recently [TLS paper, the others too?]. (XXX are we lame and: after the initial qualitative ad-hoc analysis here, we review possibilities for more analysis in the future with some of the metrics proposed in the literature -- or can we rock and already apply the techniques here?)

The rest of the article is organized as follows: As background information, we first we give a brief overview of the networked entity system in question, and of the implementation of it in the Tundra SDK. Also, we review existing techniques for API usability research and API complexity analysis. (also?: networked programming libs? alternative models too such as the messaging in sirikata? -- suggest same analysis for sirikata data (kata pong!) as future work?).

As the contribution from this research, a set of example projects are studied. Conclusions are drawn from the observations both to evaluate the conceptual level entity model, and to identify possible improvements and desireable support functionality to be developed in the surrounding platform.

Background and related work

The Entity-Component-Action (ECA) model

- entity system desc (copy-paste from original draft, short, refer to IEEE IC)
    - nice new diagram perhap, to summarize ECA+A

Tundra SDK overview

- what is there besides entities in the Tundra API -- and what is Tundra overall (again hopefully copy-paste from 1st draft)
- other platforms -- Unity3D having the same model etc?

API complexity analysis / research

(or: research methodology?)

Motivation:

API complexity has been analyzed to "(increase) the failure proneness of source code" in a recent exploratory study [cmu-api_failures]. Also, it is well noted how making good APIs is hard -- creating a bad one is easy [api-matters].

- of software complexity and api analysis research, techniques, metrix
    - can we apply here, what and how?

In this paper, we examined the impact of API complexity on the failure proneness of source code files using data from two large-scale systems from two distinct software companies and nine open source projects from the GNOME community. Our analyses showed that increases in the complexity of APIs are associated with in- creases in the failure proneness of source code" [cmu-api_failures]

Tundra game projects dissected

- pong and chesapeake / plankton case still
    - concepts of local/remote
        - physics authorativity in Pong
        - the whole net load avoiding, described with the diag in plankton case
- circus -- what about it? there is gameobject.js and gamestate.js
- mixed reality city game, with websockets too
    - an example where the extensibility seems to work (Q: is there state sync, what does 'addPolice' action actually do etc? - how is the Graffity info in the system?
- others?

Results

...

References

api-matters Michi Henning, API Design Matters, Communications of the ACM Vol. 52 No. 5 http://cacm.acm.org/magazines/2009/5/24646-api-design-matters/fulltext

cmu-api_failures(1, 2, 3) Marcelo Cataldo1, Cleidson R.B. de Souza2 (2011). The Impact of API Complexity on Failures: An Empirical Analysis of Proprietary and Open Source Software Systems. http://reports-archive.adm.cs.cmu.edu/anon/isr2011/CMU-ISR-11-106.pdf

api-complexity-analysis Comparing Complexity of API Designs: An Exploratory Experiment on DSL-based Framework Integration. http://www.sba-research.org/wp-content/uploads/publications/gpce11.pdf