С. М. Абрамов, Н. С. Живчикова, С. В. Знаменский, Е. С. Иванов, А. В. Котомин, Д. Н. Степанов, Е. В. Титова, В. Н. Юмагужина

Архитектура системы для разработки технологий организации сложной совместной деятельности[1]

В данной статье авторами проанализированы современные подходы к организации совместной деятельности (в частности, разработке и изменения контента) с применением информационных технологий и недостатки этих подходов. Предложена новая модель организации совместной деятельности, ориентированная на стандарты управления качеством.

Постановка задачи

Стандарты управления качеством ISO 9001:2008, ГОСТ Р ИСО 15489 предусматривают регулярные аудит и улучшение бизнес-процессов. Эта деятельность, по сути, тождественна той, что осуществляется в ходе предпроектного обследования при создании информационной системы и может в любой момент привести к осознанию потребности существенных изменений имеющейся системы.

Осознанная потребность изменений существующей структуры чаще всего остается неудовлетворенной, поскольку разработать новую систему, как правило, оказывается дешевле, чем внести изменения в старую. Обычно при этом старые данные еще долго остаются полезными, и приходится одновременно поддерживать в рабочем состоянии несколько информационных систем, что создает значительные неудобства и заметно снижает качество сервисов.

Основным препятствием к созданию гибко перестраиваемых систем, на наш взгляд, является общепринятое и закрепленное во всех учебниках представление об информационной системе как о товаре, отчуждаемом от производителя и покупаемом потребителем. Это упрощает отношения между разработчиками и потребителями до отношений продавца и покупателя, и производитель становится объективно заинтересованным в создании негибких систем с необоснованно декларированными возможностями адаптивного учета изменений в бизнес-процессе.

Во всех учебниках по информационным системам и базам данных (которые финансируют производители) ошибочно предполагается очевидным, что в ходе предпроектного обследования бизнес-процессы могут быть изучены настолько, чтобы созданная на этой основе система могла длительное время эффективно эксплуатироваться без активной поддержки разработчиков.

Пользователь в роли разработчика

Сложной совместной деятельностью будем называть многопользовательскую активность по созданию и развитию информационного контента со следующими особенностями:

1. Структура контента и организация многопользовательского доступа могут потребовать непредсказуемых изменений.
2. В любой момент может потребоваться доступ к прежнему состоянию каждой части контента.

Известные описания «эволюционирующих» или «устойчиво развивающихся» информационных систем, адресованные этой

[1] При поддержке РФФИ, грант № 09-07-00407-а.

проблеме [11,13], относятся к первому пункту, но не предусматривают второго.

Речь в них идет отнюдь не о немыслимом интеллекте информационной системы, гениально угадывающей потребности пользователей и гибко подстраивающейся под них, а о постоянном органичном участии программистов в творческом процессе. Такая система не может быть сделана под заказ, она может быть только выращена в той среде, где она будет использоваться, и в этом процессе часть коллектива пользователей должна превратиться в команду сопровождающих программистов.

Вместе с тем второй пункт крайне важен. Как правило, перестраивать систему без временной потери доступа к старым данным не удается. Без заложенного в основу полноценного доступа к прежним состояниям системы невозможно избежать запутывающего пользователей одновременного использования нескольких версий системы.

Острая потребность в сложной совместной деятельности ощущается в некоторых сферах управления социальными процессами, например, при разработке инновационных технологий организации обучения. При этом полезны такие тесно взаимосвязанные инструменты совместной деятельности, как индикация изменившихся данных и важных дел, требующих участия пользователя, интерфейсы совместного доступа к данным и совместной выработки решений, алгоритмы персонализации информации и автоматического учета активности, продуктивности и дисциплинированности каждого участника, включая разработчиков системы. Эти инструменты очевидно не могут создаваться вне системы или независимо друг от друга и способны дать значимый эффект лишь в комплексе.

Задача создания столь необычной системы ранее не рассматривалась, вероятно, по причине особой сложности и высокой затратности начального этапа разработки, который не сможет опереться на распространенные стандарты и технологии проектирования информационных систем. Однако в случае успешного решения результат мог бы оказаться полезным и в других сферах.

Важнейшая из сфер сложной совместной деятельности — это разработка сложного программного обеспечения. Информационная система, обещающая повышение эффективности творческого сотрудничества с участием разработчиков, очевидно, обязана стать идеальной интегрированной средой для ее (этой системы) дальнейшей разработки и сопровождения.

Сохранение истории и авторства

Первые попытки организации хранения полной истории разработки были связаны с задачами управления версиями программ и решались многочисленными системами управления ревизиями исходных кодов. Этот подход быстро распространился на сопроводительную документацию и другие документы. Возможность анализа изменений и возврата к прежней версии стала основой функциональности многочисленных wiki-сайтов, включая Википедию.

Средства отслеживания изменений стали закладываться и в основу корпоративных систем для обеспечения сохранности информации и безопасности. Например, первая в России биллинговая система с открытыми исходными кодами Nadmin [4] автоматически сохраняет вместе с измененными данными дату и время изменений и аутентифицирующие данные.

Этот подход отличается от идеи темпоральных (и битемпоральных) реляционных баз данных. В реляционных базах время — это один из типов данных, доступ к которому ограничивается так же, как и к другим данным, а проблема состоит в эффективной организации доступа [9, 14].

В рассматриваемой постановке время сохранения (и попутно авторство), согласно стандартам контроля качества, должны фиксироваться на системном уровне без возможности фальсификации программистами. В ситуации пользователей с ролью разработчика это первая из основных мер

безопасности. Следовательно, транзакционное время должно фиксироваться на более низком уровне, и технологии битемпоральных реляционных баз данных напрямую использованы быть не могут.

Другой принципиальный недостаток реляционной модели данных состоит в том, что она абстрагируется от учета продолжительности транзакции, что отрывает модель от моделируемой реальности, в которой транзакция имеет непродолжительное неопределенное состояние. Это объективно препятствует построению высококачественного сервиса на основе реляционных баз данных.

Отслеживание изменений

В информационной системе с качественным контролем управление версиями должно интегрироваться с эффективными механизмами организации доступа. В wiki-системах проработаны механизмы совместного создания и редактирования документов, но слабы средства организации экспертизы и утверждения согласованной информации. Часто подобные механизмы надстраиваются над wiki-системами, оказываясь неудобными и малоэффективными.

Пятилетний опыт интенсивной эксплуатации системы Twiki [10,15], все эти годы остававшейся в списке лидеров wiki-систем, выявил полную непригодность заложенной модели структурирования информации и разграничения доступа для построения на ее основе качественной информационной системы.

Главной проблемой стало оповещение эксперта, руководителя (или любого пользователя) о появлении ожидающих срочного рассмотрения или утверждения документов или поправок.

Здесь возникли неожиданные сложности. Если право поставить визу имеет любой из группы пользователей, то все они должны видеть необходимость этого действия до тех пор, пока кто-то из них ее не поставит. Список текущих дел (или задач) пользователя (назначенных к исполнению им самим или кем-либо еще) должен динамически отражать изменения их статуса.

Сообщения о системных неполадках и жалобы пользователей полезно оперативно и гарантированно доводить до соответствующих администраторов и разработчиков системы, причем любой из них должен иметь возможность выключить тревожный сигнал.

Оповещение о деле не должно отвлекать от работы, никак с ним не связанной. Его видимость должна учитывать близость к текущей деятельности. У пользователя должны быть возможности контекстно влиять на видимость оповещений.

Значительно усиливает трудность задачи организации динамического оповещения пользователя об изменениях в данных необходимость учета возможных изменений текущей организации доступа, настроек и текущей активности пользователя, отражающей его сиюминутные интересы.

Проблема в том, что обычное сохранение данных одним пользователем может оказаться настолько важным для других, что их полезно известить об этом, не дожидаясь, пока они закончат чтение сложной страницы или заполнение большого текстового поля, родственного этим измененным данным. Другая ситуация — практически одновременное изменение общих данных двумя пользователями. Если не сообщить о возникшем конфликте, ценная информация может быть утеряна.

Стандартный подход передачи сообщений через рассылку неприемлем по двум причинам:

1. Если опасность уже миновала, то внимание к ней привлекать излишне, поэтому сообщения можно ликвидировать, но стандартные средства такой возможности не дают.

2. В случае изменения роли пользователя он должен сразу же увидеть ситуацию по-новому, часть сообщений должна исчезнуть, а другая появиться.

Поиск в литературе и в сети Интернет разумных подходов и ссылок на программ-

ные средства для решения этой проблемы оказался безуспешным.

Фундаментальные особенности системы

Рассматриваемая модель построения информационных систем существенно отличается от доминирующей MVC [12] и подавляющего большинства других «рациональных» моделей информационных систем: в ней организация хранения и обработки данных принципиально не может определяться фиксированной бизнес-логикой, извлеченной из предпроектного обследования. Архитектура системы должна соответствовать логике эффективного обеспечения взаимодействия сотрудников, информирования их о важных изменениях, а не тем, на какие изменения в данных направлена их совместная деятельность.

Остается один ориентир: возможность создания эффективного и дружественного пользовательского интерфейса. Рассмотрим требуемые качества, отсутствующие в большинстве существующих систем.

Контекстная автономность

Первое необходимое качество — это возможность любую разумно выделяемую часть системы перестроить с ее бизнес-логики на любую другую без приостановки сервиса.

Основным содержанием обсуждаемой системы является слабо структурированная информация (данные и исполняемый код). Негативный опыт использования абстрактных идентификаторов данных привел к такой организации информации, в которой каждый элемент идентифицируется строкой, составляющие которой несут семантическую нагрузку.

Будем считать контекстом совокупность данных с фиксированным началом строки идентификатора. Примером контекста является содержимое директории в файловой системе или на статическом сайте (идентификаторами выступают полные пути к файлам).

Назовем информационную структуру контекстно-автономной, если организация доступа к данным любого контекста (и, разумеется, сами данные) могут быть изменены независимо от остальной части системы без приостановки сервисов, в которых контекст не участвует.

Контекстная автономность характеризует гибкость структуры системы, без которой невозможно четко разграничить доступ пользователей-разработчиков системы [2,3].

Сохранение истории изменений

Поскольку перестройка структуры информации обычно сопровождается трудновосполнимыми потерями доступа к данным, то контекстная автономность всегда должна дополняться специфическими механизмами безопасности. Сохранение полной истории изменений естественно сопроводить доступом к любым ранее доступным интерфейсам с ранее актуальными данными.

В системе необходимо хранить все версии данных и исполняемого кода (и рабочие ссылки на них) и при обработке запросов с указанием времени запроса должна вызываться действительная на момент запроса версия. Это гарантирует полный доступ на чтение к прежним состояниям системы. Доступ должен быть авторизован в настоящем времени и не может фальсифицировать историю.

Хранение неактуальных версий файлов и другой информации связано с затратой ресурсов и порождает проблему освобождения от лишней информации, которая по объему может превышать актуальную в десятки и даже в сотни раз при хранении полной истории изменений как первичных, так и обработанных данных.

На самом деле проблема не так сложна, как кажется на первый взгляд. Надо отметить, что историю первичных данных хранят всегда — это логи базы данных, без которых не-

возможно восстановить ее рабочее состояние в случае поломки. Кроме этого, обычно хранится одновременно много резервных копий базы, поскольку потеря данных может быть замечена не сразу. Если информация в базе только накапливается, но не заменяется и не удаляется, то хранение более чем одной-двух копий становится избыточным.

Вместе с тем, длительное хранение ненужной информации лишено смысла. Информация, использовавшаяся давно и в течение короткого времени подлежит автоматическому удалению, и эту часть данных система может выделить и удалить без участия человека с минимальным ущербом для целостности истории.

Задача решается в фоновом режиме выделением «белых пятен истории» (промежутков времени, история которых недоступна) и удалением информации, актуальной лишь в выделенных временных промежутках. Предварительно для всех контекстов устанавливаются допустимые суммарные меры белых пятен истории (мера отрезка [a, b] равна $1/b - 1/a$), а в ночное время (совпадающее с периодом минимальной загруженности) набор белых пятен каждого контекста расширяется так, чтобы данных, актуальных за пределами белых пятен, осталось как можно меньше. Все это может делать фоновый процесс, не взаимодействующий с сервером. Так можно организовать безопасную чистку базы в периоды минимальной загруженности.

Использование истории изменений

На странице любого пользователя может присутствовать графический интерфейс («панель машины времени») для просмотра состояния URL на произвольный момент времени (от момента создания до текущего времени). Пользователь всегда видит на странице то время, с которым в данный момент работает. Он может останавливать ход времени и вновь его запускать. Перемещение во времени может осуществляться посредством перестановки флажков на временных шкалах (год, месяц, число, час, минута, секунда). Течение времени можно замедлить или ускорить, указав соответствующий коэффициент ускорения.

Такой просмотр (ускоренный, замедленный или в режиме реального времени) истории информационно наполненной страницы с цветными диаграммами и выделением проблемных участков даст беспрецедентно наглядное и точное представление о динамике происходивших в данных изменений.

Текущее время, с которым работает пользователь, передается на сервер в периодических асинхронных запросах, содержащих идентификаторы фрагментов страницы и изменения в данных.

Последствия внесения изменений в данные через машину времени опасны «эффектом бабочки». Поэтому ограничения на доступ к ним должны быть особо жесткими, а при сохранении изменений необходимо одновременно фиксировать оба времени — экранное (прошлое) и реальное.

Качественная реализация отслеживания изменений в основе информационной системы должна предусматривать возможность отмены действий пользователя (подобную Undo в офисных приложениях) и быструю ликвидацию системным администратором ближайших последствий несанкционированного доступа или возврат к предыдущей версии исполняемого кода.

Неблокирующая поточная обработка данных

Пользователю удобно иметь оперативный доступ к корректировке сопутствующей информации. Но с увеличением количества содержащихся на странице web-интерфейса редактируемых данных возрастает вероятность потери никем не замеченных изменений, сохраненных одним пользователем во время работы другого над этой же страницей.

Стандартный подход к исключению потерь информации — это использование транзакций и блокировок. Например, если два пользователя одновременно пытаются перевести по рублю: один со счета А на счет Б, а другой со счета Б на счет А, то стандартный для реляционных баз данных механизм подразумевает сохранение в логах запросов от каждого пользователя и блокировку обоих счетов при выполнении каждого из переводов. Если два разных процесса-обработчика одновременно сначала блокируют счет отправителя, а затем счет получателя, то получается вечная блокировка deadlock, наличие которой снижает производительность системы. Чем сложнее обработка, тем труднее своевременно выявлять и распутывать такие ситуации в автоматическом режиме. При сложных обработках для восстановления системы порой неизбежен перезапуск сервиса с потерей последних изменений.

Этот подход является объектом критики сторонников так называемых постреляционных баз данных, считающих излишним использование блокировок на основании маловероятности конфликта. Такой «оптимистический» подход к разрешению коллизий вероятно хорош для компьютерных игр, но, на наш взгляд, неприемлем для серьезных приложений.

Вместо тормозящих блокировок или слепого накладывания одних изменений на другие в целях разрешения конфликта, предлагается немедленно передавать каждое законченное изменение маленького фрагмента информации на сервер с оперативной индикацией изменений (или, по меньшей мере, их наличия) у всех пользователей, видящих форму для изменения этих данных. Тогда можно будет увидеть, какая часть информации относительно стабильна, а какая только что была изменена, где другой пользователь сейчас правит наши данные, а где его правки давно наложились на наши. Если в подобных ситуациях возникает конфликт, то разрешать его должны пользователи, а роль системы — только указать его возможное наличие.

Для неблокирующего разрешения обоснованных конфликтов полезны доступность истории изменений каждого фрагмента информации, контекстный чат либо форум для обсуждения спорных ситуаций и разбиение большого документа на независимо изменяемые фрагменты.

В случае, когда (как в примере с переводом денег) для конфликта нет основы, в совместных блокировках различных данных (а именно они порождают проблему) никакой необходимости нет. Пользователи могут одновременно взять по рублю с одних счетов и положить на другие. При этом никто не пострадает, если гарантируется, что взятый рубль будет положен и в момент, когда деньги взяты, но еще не переложены, не произойдет вычисление суммы средств на счетах или другой операции, требующей согласованности этих данных.

Реляционные базы данных были придуманы для того, чтобы программист мог сосредоточиться на логике запросов, не думая о необходимости синхронизации. В них состояние базы всегда определено и консистентно. Однако при распределенном хранении и многопоточной обработке данных затраты на синхронизацию становятся основной составляющей используемых ресурсов. Поэтому строить эффективно масштабируемую систему, подразумевая все данные в любой момент строго согласованными, становится бесперспективно.

Незначительная на первый взгляд особенность взаимодействия в корне меняет базовые требования к организации хранения, обработки и визуализации данных. Сведения о счетах в базе, как и их отображение на экране браузера, не могут отражать состояние именно на текущий момент. Разумное запаздывание неизбежно. Пользователю вполне достаточно, чтобы в любой момент времени каждая часть экрана отражала состояние, в котором соответствующие данные находились в близкие моменты времени. Разница незаметна при близости в десятые доли секунды и не раздражает, даже составляя секунды, если она естест-

венно отражает перегруженность сервера или сложность вычислений.

Аналогично и функция, обрабатывающая данные информационной системы (например, вычисляющая сумму средств на счету), не должна выполняться, пока все входные значения не окажутся в стабильном состоянии. Если система может гарантировать оперативную актуализацию входных данных, то о последнем можно судить по давности их изменений. Но она в любом случае может фиксировать информацию о транзакциях (как в обычных логах) и вместо блокировки данных откладывать выполнение функции. При перегрузках сервиса это грозит тем, что сводные данные на экране пользователя будут немного отставать от первичных, но такая потеря качества обслуживания намного лучше, чем перебои в сервисе, особенно если на экране указано, насколько устарели сводные данные.

Стандартные для реляционных СУБД строгие требования ACID к безупречности логических взаимосвязей всех данных в любой момент времени не столь непреложны для динамически меняющейся информации [7] и заведомо игнорируются известными поисковыми машинами, популярности которых не вредит тот факт, что они не затрудняют себя постоянной перепроверкой всех найденных ссылок.

Важную роль в оперативном доведении до пользователя информации имеет асинхронное ее обновление, при котором обновления на экране регулярно происходят даже при пассивности пользователя. Избежать хаотических дерганий страницы web-интерфейса из-за незамедлительного показа изменений при этом нетрудно. Достаточно разбить страницу на фрагменты и отображать значки, сигнализирующие о наличии непросмотренных изменений.

Динамическая навигация

Пользователь, одновременно включенный в сотни совместных дел, должен видеть, какие из них предполагают или ожидают его немедленного участия. Если, например, где-то изменено сохраненное им значение, для него может быть важно, не откладывая, разобраться с конфликтной ситуацией. Подобных ситуаций бывает много, и они могут в разной степени волновать пользователя и настаивать на его участии.

В любом состоянии прокрутки на экране должна быть видна иконка, отражающая важность не просмотренных пользователем изменений в его делах с учетом его персональных настроек. Эта иконка будет открывать оптимизированное под текущую ситуацию контекстное меню с наиболее важными для пользователя делами, близкими к контексту настоящего момента времени.

Средства реализации

Попытки частичного сочетания рассмотренной функциональности в системе поддержки проведения научных конференций отчетливо обозначили проблему производительности: выявление обновлений системы и доведение их до пользователей вылилось в обработку больших объемов рассредоточенных данных.

Обоснованные сомнения в возможности достижения требуемой производительности при использовании Microsoft SQL Server привели к в выводу о целесообразности опоры на Oracle Berkeley DB или Tokyo Cabinet, обещающих в десятки раз более высокую производительность [8].

В качестве языка реализации был выбран Perl. Арсенал этого языка оказался достаточно богатым, чтобы обеспечить многократное использование хранящегося в базе данных кода без повторных загрузок и без конфликтов между разными загруженными версиями одного модуля.

Архитектура и особенности реализации

Поскольку принципы и средства информационной поддержки должны уточняться и прорабатываться в ходе эксплуатации системы, то она базируется на идеях эволюци-

онного прототипирования [9] и должна стать удобной интегрированной средой для саморазработки.

Клиентская часть системы представляет собой обычный современный браузер с включенной поддержкой javascript (допускается кратковременная работа без javascript).

Клиентские скрипты выполняют следующие функции:

- регулярно (несколько раз в секунду при измененных данных, раз в несколько секунд при длительном бездействии пользователя) пересылают на сервер асинхронный запрос, содержащий:
  - идентификаторы фрагментов страницы;
  - изменения в данных, не фиксированные в базе серверных данных;
  - текущее время, если используется панель машины времени;
- получив в ответе обновления, показывают их на странице без ее полной перезагрузки;
- получив хеш последних записанных пользователем данных, зафиксированных на сервере в течение последней минуты, сверяют их с посылаемыми серверу и удаляют принятые сервером из списка отправляемых.

Компоненты серверной части

Загрузчик анализирует запрос и запускает код основной процедуры обработки конкретного запроса.

Его задачей является разбор параметров запроса к серверу и обращение к базе авторизации, возвращающей имя той версии программного модуля из каталога, которую загрузчик должен вызвать.

Простота и неизменность кода загрузчика обеспечивают не только ретроспективную обработку с кодом и данными, актуальными на указанный момент времени, но и рабочее тестирование в той же системе следующей версии кода. Это облегчает и делает безопасными частые обновления системы и позволяет параллельно прорабатывать несколько веток кода.

Репозиторий приложенных файлов хранит всю бинарную информацию. Это статическая директория, файлы которой доступны в сети Интернет по именам, а листинг недоступен. Имена файлов содержат дайджест содержимого, что исключает несанкционированный доступ через угадывание имени.

Хранилище текстовой и структурообразующей информации содержит текущее состояние и всю историю системы. Оно эффективно обрабатывает ретроспективные запросы с учетом пользовательских Undo и учитывает записанные при тестировании данные только в тестовых запросах.

Базовая структура данных

Все данные, включая шаблоны и имена файлов исполняемого кода, логически хранятся в базе типа BTree. Это означает, что записи в базе состоят из ключа и значения некоторого данного. Ключи записей логически образуют лексикографически упорядоченный список со сбалансированным двоичным деревом доступа к нему, позволяющим найти по строке первую запись, лексикографически превосходящую строку запроса.

Ключ записи является конкатенацией строк, среди которых обычно есть:

- префикс, идентифицирующий данное;
- строка отсчета времени в секундах с заданной для контекста точностью;
- строка отсчета реального времени в секундах и (или) идентификатор пользователя или процесса, сохранившего запись.

Это позволяет очень быстро, за считанные (логарифмически растущие с числом записей в базе) микросекунды, получить актуальное на любой заданный момент времени значение любого данного и гарантирует неизменность сохраненных данных. Значение данного является строкой, в которой может быть сериализована любая структура или код в Perl, ключ другой записи или имя файла с данными.

Значением может быть и директива пользовательского Undo — отмены последних изменений. Эта директива делает неактуальными все пользовательские изменения за период от указанного в ней момента времени до момента выдачи директивы и все их последствия. Среди таких изменений могут быть и директивы Undo. Так обеспечивается возможность отменить неудачные отмены «по горячим следам». Отмена изменений убирает все пометки о наличии и авторстве изменений, а также их последствия.

Ключ записи может содержать тестовую пометку. При тестировании все изменения в системе учитываются, но помечаются как тестовые и не отражаются на работе обычного пользователя, поскольку при обработке обычных запросов тестовые данные игнорируются. Это напоминает слои изображения в графическом редакторе, причем слой тестовых данных как бы находится под обычными слоями информации и невидим для пользователей.

Идентифицирующий данные префикс обычно является составным и формируется так, чтобы максимально использовать быстрый механизм сбалансированного двоичного дерева для проведения наследования данных при авторизации доступа к страницам и выборе нужной версии шаблона.

Рабочий цикл серверного процесса

При получении запроса серверный процесс аутентифицирует пользователя и по командной строке определяет, относится ли запрос к рабочей системе или к тестированию нового кода в ней. Если идет тестирование, то проверяется наличие в настройках пользователя указаний на особую версию системы и подгружается либо активизируется соответствующая версия.

Процесс выдает подготовленную и содержащуюся в базе информацию и пытается сохранить полученные от пользователей несохраненные данные. В случае неудачи (ресурс занят другими серверными процессами) попытка сохранения повторяется после обработки каждого следующего запроса.

Вся обработка первичных данных производится в фоновом режиме. В ключе сохраняемых результатов фиксируются время актуальности первичных данных и время окончания обработки.

Заключение

Описанная система (http://edu.botik.ru) разрабатывается в Учебно-научном центре ИПС имени А. К. Айламазяна РАН и УГП имени А. К. Айламазяна в г. Переславле-Залесском. В 2002 году центр объединил академический институт и негосударственное высшее учебное заведение — Университет города Переславля, созданный в 1993 году. В университете обучается около 400 студентов по трем специальностям, связанным с информационными технологиями. В связи с этим УГП старается использовать информационные технологии непосредственно в учебном процессе. С использованием описываемой системы с 2007 года систематически проводятся конференции с перекрестным рецензированием [1,5,6] и с 2008 года ведется трекинг студенческих проектов (рис. 1).

В процессе разработки системы в числе прочих ведутся исследования по влиянию внедрения информационных технологий в учебный процесс на познавательную активность студентов. Внедрение модуля поддержки проведения научных конференций позволило в 3 раза увеличить число участников студенческой научной конференции. Модуль дает возможность студентам наравне с оргкомитетом рецензировать и оценивать представленные на конференцию статьи, проводить их доработку (рис. 2).

Прозрачная система голосования вносит элемент соревнования в проведение конференции, что существенно повышает интерес и студентов, и преподавателей.

Система ежегодно перерабатывается с сохранением функциональности. В 2009 году она была перестроена на основе рас-

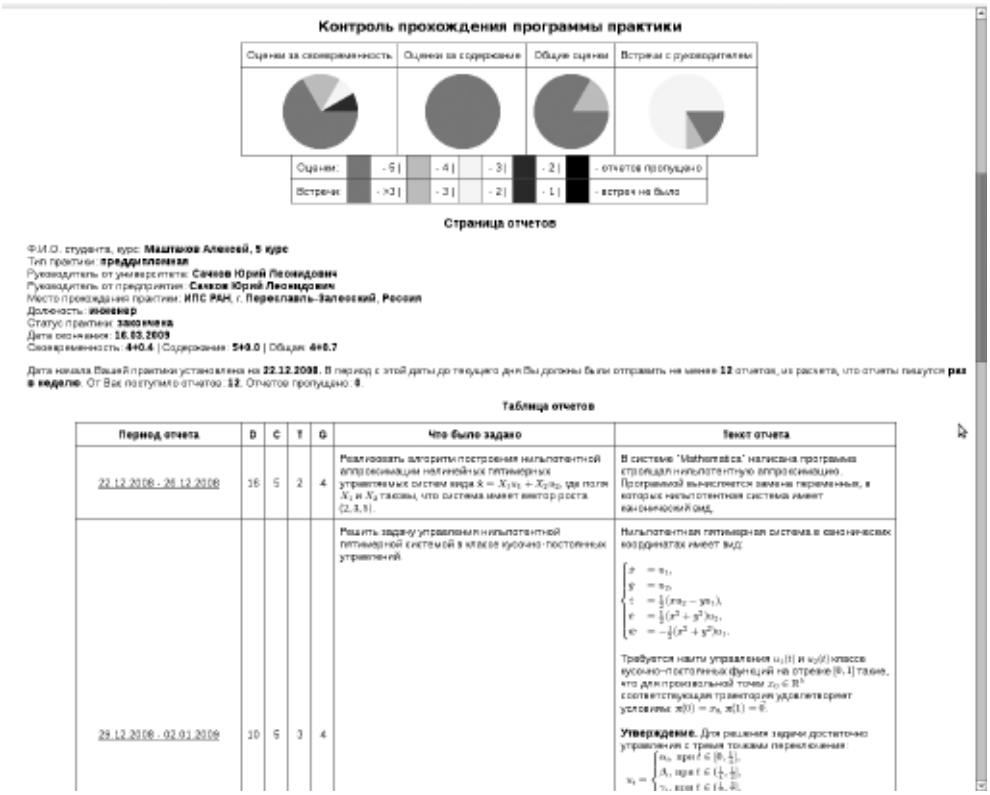

Рис. 1. Пользовательский интерфейс системы трекинга проектов

смотренных принципов с целью достижения нового качества организации учебного процесса.

Список литературы

1. *Амелькин С. А., Знаменский С. В.* Информационная поддержка организации сложной совместной деятельности. // Труды международной конференции «Программные системы: теория и приложения», ИПС имени А. К. Айламазяна РАН, г. Переславль-Залесский, май 2009. Переславль-Залесский: Изд-во «Университет города Переславля», Т. 1, 2009.
2. *Живчикова Н. С., Титова Е. В.* Логическая модель изменчивых организационных структур. // Тезисы международной конференции «Системы проектирования, технологической подготовки производства и управления этапами жизненного цикла промышленного продукта (CAD/CAM/PDM — 2008)». М: Институт проблем управления РАН, 2008.
3. *Знаменский С. В.* Хорошо масштабируемое автономное администрирование доступа. // Международная конференция «Программные системы: теория и приложения», Переславль-Залесский, октябрь 2006, Наука,-Физматлит, М. Т. 1, 2006.
4. *Карлаш А. В., Абрамов С. М., Жбанов П. Г., Нестеров А. С., Ермилова Е. В., Шевчук Ю. В.* Надмин — административно-расчетная система для региональных сетей. // Труды Всероссийской научной конференции « Научный сервис в сети Интернет «, 20–25 сентября 2004 г. Новороссийск, Изд-во МГУ, 2004.
5. *Коряка Ф. А.* Автоматизированная система управления вузом — UPIS. // XI научно-практическая конференция «Университета г. Переславля». Переславль-Залесский, апрель 2007, изд.-во «Университет города Переславля», Т. 1, 2007.
6. *Степанов Д. Н.* Алгоритм назначения рецензентов как часть проведения научных конферен-

Рис. 2. Перекрестное рецензирование и совместная работа над публикацией

ций при поддержке информационной системы UPIS. // XII научно-практическая конференция Университета г. Переславля. Переславль-Залесский, апрель 2008, изд.-во «Университет города Переславля», Переславль-Залесский. Т. 1, 2008.

7. ACID — // Материал из Википедии — свободной энциклопедии, эл. ресурс, http://ru.wikipedia.org/wiki/ACID.
8. Berkeley DB // Материал из Википедии — свободной энциклопедии, эл. ресурс: http://ru.wikipedia.org/wiki/Berkeley_DB.
9. *Gubiani D., Montanari A.* A Relational Encoding of a Conceptual Model with Multiple Temporal Dimensions. // Lecture Notes in Computer Science Volume 5690/2009. BookDatabase and Expert Systems Applications, 2009.
10. Foswiki — The free and open source enterprise wiki. 2008–2009, эл. ресурс: http://www.foswiki.org/.
11. *Mart Roost, Karin Rava, Tarmo Veskioja,* Supporting self-development in service oriented information systems, // Proceedings of the 7th Conference on 7th WSEAS International onference on Applied Informatics and Communications, p. 52–57, August 24–26, 2007, Vouliagmeni, Athens, Greece.
12. Model-View-Controller (MVC) // Материал из Википедии — свободной энциклопедии, эл. ресурс: http://ru.wikipedia.org/wiki/MVC.
13. *Schmidt Nils-Holger, Erek Koray, Kolbe Lutz M., Zarnekow Rüdiger.* Sustainable Information Systems Management. // Business & Information Systems Engineering Volume 1, Number 5 / October, 2009.
14. *Stantic Bela, Sattar Abdul and Terenziani. Paolo.* The POINT approach to represent now in bitemporal databases Journal of Intelligent Information Systems Volume 32, Number 3 / June, 2009.
15. TWiki — the Open Source Enterprise Wiki and Web 2.0 Application Platform by Peter Toeny and contributing authors, 1998–2009, эл. ресурс: http://www.twiki.org.