УДК 004.4

Т.К. СКРИПНИК, Е.А. МАНЗЮК, С.О. СВИСТУН
Хмельницький національний університет

АРХІТЕКТУРА КАРКАСУ MODEL-VIEW-CONTROLLER ПРИ РОЗРОБЦІ ВЕБ-ДОДАТКІВ

У роботі розглянуто патерни проектування MODEL-VIEW-CONTROLLER (MVC) та MODEL-VIEW-VIEWMODEL (MVVM), способи взаємодії і ролі кожного компонента архітектурних каркасів, можливості застосування сервісів при розробці програми. Наведено діаграми http-запит-відповідь при використанні сервісів і без них. Описано основні можливості фреймворка AngularJS. Показана структура взаємодії компонентів веб-додатку при використанні фреймворка.

Ключові слова: MVC, AngularJS, JavaScript, сервіс, патерн, веб-додаток.

T.K. SKRYPNYK, E.A. MANZIUK, S.O. SVISTUN
Khmelnytsky National University

MVC SOFTWARE ARCHITECTURE PATTERN FOR WEB-APPLICATION

Web application - a special kind of software that is available to users over the network that uses the browser as a client. The main advantages of web applications include cross-platform interoperability and the ability to update and maintain the application without installing software on client devices. The main criteria in choosing web application architecture – possibility of modular development, maintenance, documentation and structuring code. Design pattern Model-view-controller (MVC) provides separation of application data, user interface and control logic. To use MVC for web applications, authors used a combination of scripts and server components and common objects to implement various components of the application. As an alternative to this solution is supposed to separate connection of controller and model by such called Service. Services, acting as reusable code libraries for other program components. They ensure application functionality, such as Logging, error handling, check permissions, exchange messages and caching. Based on the foregoing analysis, we can conclude about targeted on-appropriateness of using MVC pattern in the web application. This approach will greatly simplify the modular development and further support software.

Keywords: MVC, AngularJS, JavaScript, service, pattern, web-application.

Вступ. Стрімке зростання мережі Інтернет – це, щонайменше, часткова заслуга її використання в якості інструменту особистої публікації. Спочатку дані, якими люди обмінювались по мережі, складалися в основному з статичної інформації, що зберігається в файлах. Файли можна було редагувати, але дійсно динамічних інформаційних служб було мало.

Ситуація набула якісних змін з появою динамічної мережі, що є результатом розвитку динамічних служб. З'явилися нові сервіси – від CGI скриптів для пошукових машин до пакетів програм, що сполучали веб-додатки з базами даних. Стало недостатнім створення веб-сайту – виникла необхідність проектувати веб-додаток [1].

Веб-додаток – особливий вид програмного продукту, який доступний користувачам по мережі, використовує браузер в якості клієнта і складається з набору клієнт-серверних сценаріїв, HTML-сторінок та інших ресурсів, які можуть бути розподілені між кількома серверами. Сам додаток доступний користувачам за певним шляху всередині веб-сервера.

До основних переваг веб-додатків можна віднести крос-платформенну сумісність і можливість оновлювати і підтримувати додаток без установки програмного забезпечення на клієнтські пристрої.

Постановка задачі. На початковій стадії розробки програмних додатків стоїть завдання вибору архітектури. Архітектура програмного забезпечення в традиційному сенсі включає визначення всіх модулів програм, їх ієрархії і сполучення між ними і даними.

Основними критеріями при виборі архітектури веб-додатка – можливість модульної розробки, супроводу, документації і структуризації коду.

Виклад основних матеріалів дослідження. Патерн MVVM. MVVM – це шаблон проектування додатків для розділення коду користувацького інтерфейсу і іншого коду. MVVM дозволяє декларативно визначати користувацький інтерфейс і використовувати розмітку прив'язки даних, що дає змогу зв'язати з іншими рівнями додатку, які містять логіку та дані. За допомогою прив'язки даних застосовується вільний взаємозв'язок, який синхронізує користувацький інтерфейс і пов'язані дані, а також відправляє командам вхідні дані, які ввів користувач.

Враховуючи мету зменшення працезатрат на розробку складного програмного забезпечення, припустимо, що необхідно використовувати готові уніфіковані рішення. Адже шаблонність дій полегшує комунікацію між розробниками, дозволяє посилатися на відомі конструкції, знижує кількість помилок.

З перших кроків створення користувацьких інтерфейсів програм різні шаблони спрощували роботу розробників. Наприклад, шаблон модель – подання – презентатор (MVP) був популярний на різних платформах програмування користувацьких інтерфейсів. MVP – це вид шаблонів модель-подання-контролер, який існує вже кілька десятиліть. Суть його полягає в тому, що подання потребує презентатор для заповнення даними моделі, реакції на дії користувача, надання перевірки вводу (у тому числі за рахунок передачі цієї функції моделі) та інших подібних завдань.

Якщо говорити про відмінності MVVM і MVP, то модель подання не потребує посилання на

подання. Подання прив'язується до властивостей моделі подання, яка, в свою чергу, представляє дані в об'єктах моделі та інших станах, потрібних для цього подання. Модель у даному випадку встановлюється в якості контексту подання (Data Context), тому легко створювати прив'язки між нею та поданням. Усі дані в моделі подання автоматично оновлюються через механізм прив'язки. Коли користувач натискає кнопку в поданні, для потрібної дії виконується команда в моделі подання.

Класи подання не знають про існування класів моделі, а модель подання і модель не знають про подання. Модель насправді взагалі не має уявлення про те, що існують модель подання та подання. Це дуже слабко пов'язана конструкція, і це дає ряд переваг.

MVVM – своєрідна загальноприйнята мова розробників WPF, тому що вона добре пристосована для платформ на XAML, а такі платформи створювалися для спрощення збірки застосувань за допомогою шаблону MVVM (та інших). У корпорації Майкрософт MVVM використовувався для внутрішніх цілей при розробці додатків WPF, наприклад, Microsoft Expression Blend, поки основна платформа WPF ще тільки створювалася. Багато частин WPF, наприклад, модель контролю без перегляду та шаблони даних, використовують значний розподіл показу від стану та поведінки, що застосовується в MVVM.

Одним з найважливіших моментів, який робить MVVM дуже зручним шаблоном, – це інфраструктура прив'язки даних. За рахунок механізму прив'язки властивостей подання до моделі подання виходить слабке зв'язування цих компонентів, що повністю звільняє розробника від необхідності писати в моделі подання код, який безпосередньо відповідає за оновлення подання. Дана система також підтримує перевірку допустимості введення, яка перевіряє наповнення введених даних.

Крім функцій WPF і Silverlight, за рахунок яких MVVM стає ефективним способом структурування додатку, цей шаблон популярний ще й тому, що додаток, організований відповідно йому, легко піддається модульному тестуванню. Якщо логіка взаємодії додатку знаходиться в наборі класів моделі подання, стає легко написати тестуючий код. У якомусь сенсі подання і модульні тести – різні типи споживачів моделі подання. Набір тестів для моделі подання додатка забезпечує вільне і швидке регресійне тестування, яке зменшує вартість підтримки програми в майбутньому.

Крім зручності створення автоматичних регресійних тестів, тестування класів моделі подання може допомогти правильно проектувати інтерфейси, для яких легко робити теми оформлення. Проектуючи додаток, часто треба вирішувати що помістити в подання або в модель подання, уявивши собі, чи потрібно буде писати модульний тест, який використовує модель подання. Якщо можна написати модульні тести для моделі подання, не створюючи об'єкти користувацького інтерфейсу, то можна повністю укласти модель подання в тему оформлення, так як у неї немає залежності від певних візуальних елементів.

Нарешті, для розробників, які співпрацюють з проектувальниками інтерфейсів додатків, використання MVVM полегшує створення безперервного потоку робіт розробника і проектувальника. Так як подання – не більше ніж необов'язковий споживач моделі подання, то неважко замінити існуюче подання іншим, яке б відповідало даній моделі подання. Ця проста дія дозволяє швидко створювати прототипи і оцінювати користувацькі інтерфейси, зроблені проектувальниками.

Шаблон MVVM – простий і ефективний набір рекомендацій для проектування та реалізації додатків. Він дозволяє розділяти дані, поведінку і подання, а значить, контролювати той хаос, який називається розробкою програмного забезпечення. Слід зазначити, що чітко дотримуватися тільки одного шаблону проектування – не завжди найкращий вибір. Наприклад, використовувати MVVM для розробки додатків на основі Windows Forms через властивості елементів управління Bindings. Основна мета – це відокремити подання від бізнес-логіки і логіки, яка їх пов'язує. Застосування, з одного боку, повинно легко тестуватися і підтримуватися, а з іншого бути зрозумілим для аналітиків.

Патерн MVC. Веб-додаток, на відміну від статичних HTML сторінок, може динамічно представляти контент з урахуванням параметрів запиту, дії користувача і налаштувань безпеки.

Як правило, веб-додаток складається з декількох під-додатків. Та частина додатку системи моніторингу, яку бачать звичайні користувачі володіє досить навантаженим інтерфейсом, що складається з різних модулів. Всі ці модулі мають власні дані і керуючу логіку.

При проектуванні додатків необхідно підібрати відповідні об’єкти, віднести їх до різних класів, дотримуючись розумну ступінь деталізації, визначити інтерфейс класів і встановити суттєві відносини між класами.

Патерни або шаблони проектування дають можливість повторно використовувати ті рішення, які виявилися вдалими в минулому і спрощують повторне використання вдалих проектних і архітектурних рішень. Пройшовши перевірку часових методик у вигляді патернів проектування полегшує доступ до них з боку розробників нових систем. Патерни дають розробнику можливість швидше знайти «правильний шлях», тут під патерним проектуванням розуміється опис взаємодії об'єктів і класів, адаптованих для вирішення загальної задачі проектування в конкретному контексті [2].

Архітектурний патерн Model-view-controller (MVC) передбачає поділ даних програми, призначеного для користувача інтерфейсу і керуючої логіки на три окремих компонента: Модель, Представлення та Контролер – так, що модифікація кожного компонента може здійснюватися незалежно.

Вперше поняття MVC виникло в мові Smalltalk-80 при підході до розробки багаторівневих графічних користувальницьких інтерфейсів в кінці 1970-х, початку 1980-х. Надалі патерн зарекомендував себе як вдала архітектура програмного забезпечення.

В даний час патерн MVC доступний для багатьох мов програмування і добре проявив себе в таких фреймворках, як Apache Struts для Java, Maypole для Perl і Rails для Ruby.

Хоча більшість сучасних фреймворків намагаються розвивати парадигми MVC, для кращої відповідності потребам розвитку веб-додатків, залишається один фреймворк, який дотримується класичного патерну, описаного в Smalltalk-80

Модель, як перший компонент шаблону, обробляє стан програми та може містити в собі будь-які дані, наприклад, масив координат. Даний компонент не має інформації про стан HTML або веб-серверів, його завдання – забезпечити можливість запиту стану об'єкта і визначити шляхи його зміни.

Другий компонент – Представлення, відображає призначений для користувача інтерфейс. У додатку, як правило, використовується кілька Представлень. Основна функція Представлення – запит даних з Моделі, без можливості зміни її стану. У веб-додатках на базі MVC Представлення створюється з використанням HTML, яке в кінцевому підсумку або відображається як повноцінна сторінка, або підставляється в потрібне місце на сторінці.

Всі дії користувача, які він виробляє в Представленні обробляються третім компонентом MVC - Контролером. Він отримує запит користувача (HTTP-запит), обробляє його і переводить в послідовність дій, які повинна виконати Модель. Крім того, контролер вибирає відповідний режим для обробки відповіді Моделі.

Наступна функція Контролера – вибір відповідної сторінки (Представлення), в залежності від статусу, який повертає Модель. Адже в залежності від того, чи має Модель запитувані дані чи ні, користувачеві необхідно або відобразити їх, або повідомити про помилку. Ця функція корисна не тільки при розробці великих програм з великою кількістю сторінок, але і для простих односторінкових додатків. За допомогою Контролера можна обробляти, наприклад, такі дії, як аутентифікації і управління сесіями.

MVC для веб-додатків**.** Щоб застосувати MVC для веб-додатків, автори використовували комбінацію скриптів, серверних компонентів і звичайних об'єктів для реалізації різних компонентів в рамках програми. На рис. 1 представлена діаграма запит-відповідь.

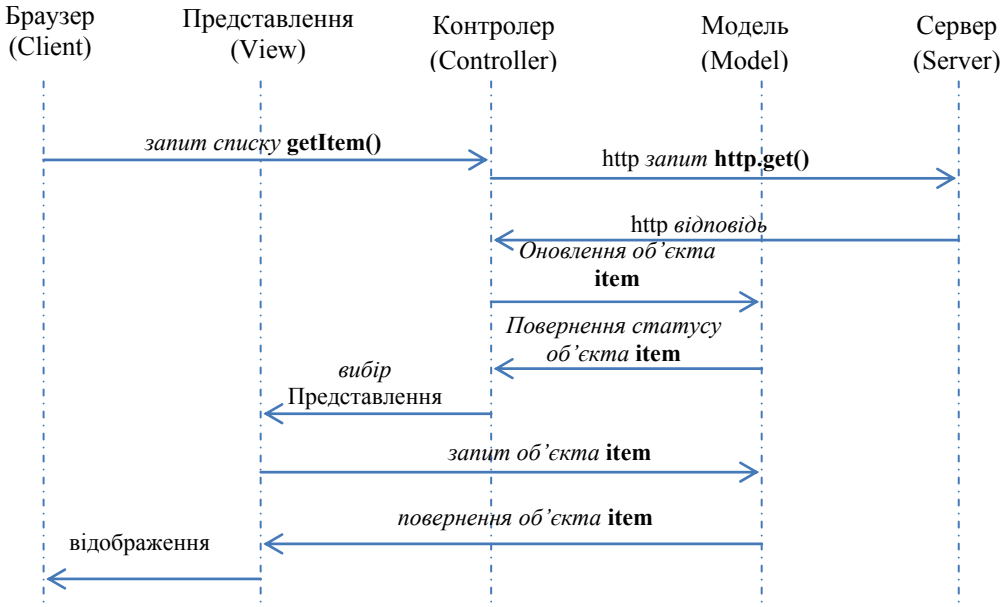

Рис. 1. Діаграма запит-відповідь

Сервіси. При розробці веб-додатку виникають ситуації, коли Модель і Контролер взаємодіють досить тісно і практично неможливо розділити код на два компонента. Для виключення цього ефекту використовують схему, при якій Контролер надає мінімальні дії, наприклад, тільки обробку HTTP-запитів і передачу результату в Модель. В цьому випадку Модель може містити в собі необхідні методи обробки результатів і форматування вихідних даних.

В якості альтернативи цьому рішенню передбачається відокремити від зв'язки Модель-Контролер так званий Сервіс.

Сервіси, виступають в ролі бібліотек багаторазового використання коду для інших компонентів програми. Вони забезпечують додаток функціоналом, наприклад, Логування, обробкою помилок, перевіркою прав доступу, обміном повідомлень і кешуванням. Сервіси можуть містити код для запиту і зберігання даних з зовнішніх серверів. Вони також можуть включати в себе функціональні можливості для сортування, фільтрації та перетворення даних в різні формати в міру необхідності [3].

На рис. 2 представлена діаграма запит-відповідь на прикладі запиту з використанням сервісу для роботи з даними.

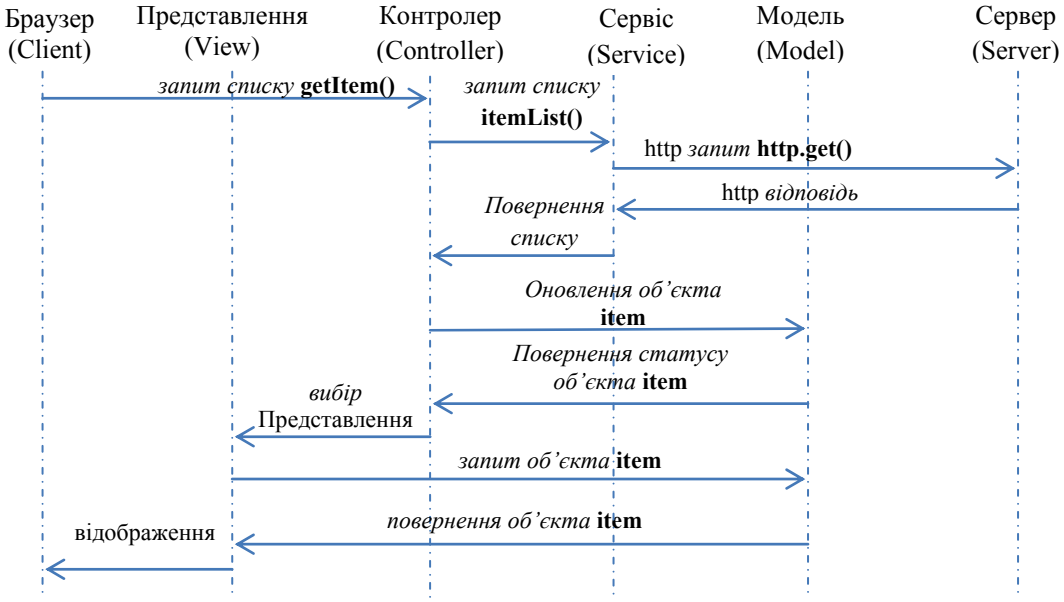

Рис. 2. Діаграма запит-відповідь за наявності Сервісу

Фреймворк AngularJS. Реалізація веб-додатку з використанням патерну MVC можлива в наступній зв'язці: як реалізації компонента Представлення можливе застосування мови розмітки HTML, компоненти Модель і Контролер реалізовані на мові JavaScript, загальний функціонал патерну забезпечує каркас або так званий фреймворк AngularJS.

AngularJS – це клієнтський фреймворк, написаний на JavaScript і виконується у браузері. Основна мета фреймворку – розширення браузерних додатків на основі MVC шаблона, а також спрощення тестування і розробки. AngularJS адаптує і розширює традиційний HTML, щоб забезпечити двосторонню прив'язку даних для динамічного контенту, що дозволяє автоматично синхронізувати Модель і Представлення. Механізм побудови Представлення не потребує явно оновлювати дерево DOM, так як фреймворк здатний слідкує за діями користувача, подіями браузера і змінами в Моделі і вчасно виявляє, коли і яке Представлення потрібно оновити [4].

AngularJS має в запасі механізм впровадження залежностей (Dependency Injection), який істотно спрощує збірку веб-додатків з невеликих, надійно протестованих служб. Загальна структура взаємодії компонентів MVC в рамках модуля програми для формування звітів з використанням даного фреймворка показана на рис. 3.

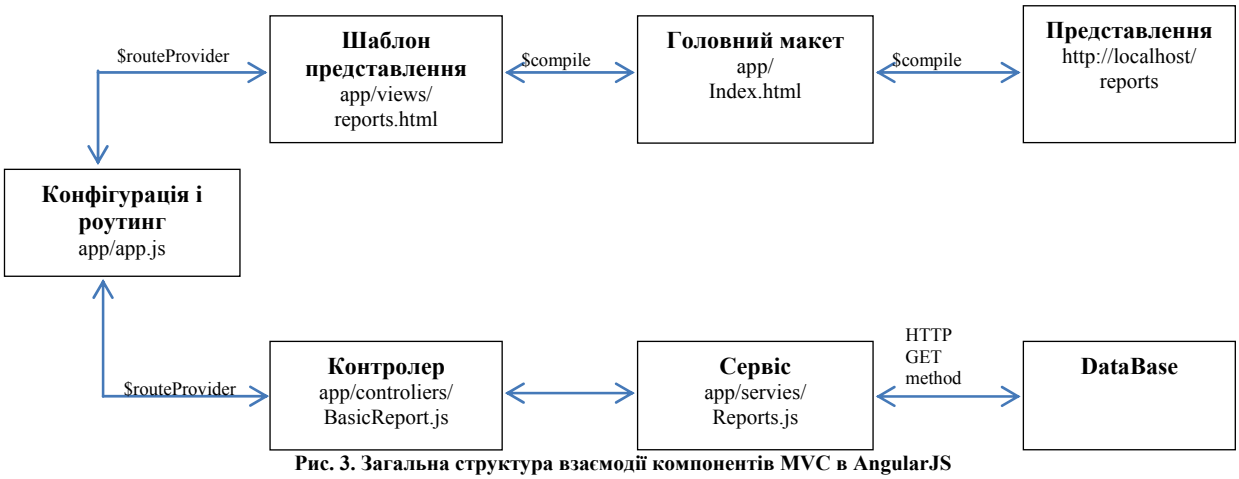

Рис. 3. Загальна структура взаємодії компонентів MVC в AngularJS

Важливим фактором є декларативний підхід AngularJS до конструювання користувальницького інтерфейсу. З практичної точки зору це означає, що Представлення описує бажаний ефект, а не способи його досягнення. Можливість декларативного опису Представлення дозволяє швидко створювати складні і інтерактивні інтерфейси. А прийняття рішень про те, коли і як змінювати елементи DOM бере на себе фреймворк.

Висновки. В роботі розвивається підхід до реалізації програмного патерну проектування MVC щодо веб-додатків. На основі викладеного аналізу можна зробити висновок, про цільових перевірок доцільність застосування патерну MVC при розробці веб-додатків. Даний підхід дозволить значно спростити модульну розробку і подальший супровід програмного продукту.

Література

1. Шкляр Л. Архитектура веб-приложений / Л. Шкляр, Р. Розен. – М. : Эксмо, 2011. – С. 55–60.
2. Гамма Э. Приемы объектно-ориентированного проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. – С. 15–44.
3. Торрес Р. Практическое руководство по проектированию и разработке пользовательского интерфейса / Р. Торрес. – М. : Вильямс, 2002. – 400 с.
4. Lavin J. AngularJS Services. Pack Publishung Ltd., Aug. 2014, ch. 1, P. 8–11.
5. Козловский П. Разработки веб-приложений с использованием AngularJS / П. Козловский, П. Дарвин. – М. : ДМК-Пресс, 2014. – С. 21–60.
6. Scott Millett Professional ASP.NET Design Patterns - Wiley Publishing, 2010. P. 8–26.

Отримана/Received : 19.4.2017 р. Надрукована/Printed :10.6.2017 р.
Рецензент: д.т.н., проф. Сорокатий Р.В.

УДК 521.1:523.2:523.6:523.9:524.6:524.8:532.5.01:534.1:621.891

Ю.П. ЗАСПА
Хмельницький національний університет, e-mail: zaspa_yuriy@ukr.net

КОНТАКТНЕ ДИНАМО ЯК ГЕНЕРАТОР КОГЕРЕНТНИХ КОСМІЧНИХ ФОРМ РУХУ ТА ДЖЕРЕЛО ПЛАНЕТАРНОЇ, СОНЯЧНОЇ, ГАЛАКТИЧНОЇ І МЕТАГАЛАКТИЧНОЇ ЕНЕРГІЇ ТА ЕЛЕКТРОМАГНЕТИЗМУ. ЧАСТИНА VII

На основі відповідного замикання системи рівнянь Максвелла наведені основні співвідношення контактної електромагнітної гідродинаміки (КЕМГД), яка заміняє собою існуючі магнітно-гідродинамічні (МГД) моделі генерації магнітного поля в космічних системах. Відзначається, що демонстративне ігнорування струму зміщення (змінного в часі електричного поля) в МГД-моделях робить їх неадекватними реальності. Розглянуті фізичні механізми контактної генерації внутрішніх гідродинамічних та електромагнітних хвиль, що характеризуються спільною фазовою швидкістю у складі когерентних структур руху. Аналізується топологія таких структур. Показано, що топологічним аналогом магнітного поля є поле завихореності контактно-наведених внутрішніх хвиль. Структура останнього зберігається внаслідок занулення сили Лоренца (і струму провідності) в окремій моді руху. Взаємний вплив різних мод в контактно-згенерованій космічній турбулентності, між тим, обумовлює дисипацію електромагнітної енергії. Наведені експериментальні результати щодо генерації електромагнітних хвиль в процесах контактної взаємодії металів в технічних системах. Спільність основних спектральних компонент акустичної та електромагнітної емісії в цих процесах прямо підтверджує запропонований механізм контактної генерації когерентних структур руху. Розглянута турбулентна трансформація таких структур в космічних системах. Показано, що розпад цих форм на периферії систем пов'язаний із зменшенням електропровідності плазми. Це пояснює розсіяння електромагнітної енергії, зокрема, в активних зонах на Сонці, на зовнішній границі геліосфери, а також потужне електромагнітне випромінювання в радіогалактиках. Відмічено, що саме електрична компонента контактно-наведеного електромагнітного поля прискорює електрони (та інші заряджені частинки) до релятивістських та ультрарелятивістських швидкостей, що в присутності крупно масштабної магнітної компоненти обумовлює відоме магнітно-гальмівне (синхротронне) випромінювання в космічних системах. Такий механізм не потребує штучних теоретичних конструкцій на кшталт чорних дір, магнітарів,темної енергії і т.п. Припускається, що зникнення магнітного поля на Місяці та Венері пов'язано з вичерпанням металізованого водню в контактних розривах ядер цих космічних об'єктів. В то й же час, наявність цього метастабільного активатора контактного динамо-процесу у відповідних розривах супутників Юпітера Іо та Ганімеда забезпечує контактну генерацію електромагнітного поля на цих супутниках. Аналізується ієрархічна взаємопов'язаність підсистем космічної турбулентності та каскадний транспорт енергії збурень між ними, що виключає застосування відомої теореми віріала до окремо взятої підсистеми та відкидає необхідність темної матерії та темної енергії, спекуляція якими є основою сучасної астрофізики та космології. Відмічено, що діапазон часових та просторових масштабів космічного контактного динамо перевищує п'ятдесят порядків величини.

Ключові слова: *контактне динамо, космічна система, когерентна структура руху, електромагнітне поле, електромагнітні хвилі, внутрішні хвилі, контактна електромагнітна гідродинаміка, магнітна гідродинаміка, рівняння Максвелла, синхротронне випромінювання, турбулентність, енергетичний каскад, металічний водень.*