Jordanian Journal of Computers and Information Technology (JJCIT), Vol. 12, No. 03, September 2026.

TOWARDS RELIABLE IOT AND WEB API MASHUPS: A HISTORY-AWARE LINKED DATA COMPOSITION APPROACH

Mohammed Amine Belfedhal[1], Mimoun Malki[2] and Abderrahmane Khiat[3]

(Received: 29-Jan.-2026, Revised: 23-Apr-2026, 25-May-2026 and 14-Jun.-2026, Accepted: 16-Jun.-2026)

ABSTRACT

The growing number of heterogeneous Web and IoT APIs poses a fundamental challenge to the reliability and reusability of modern applications, as existing API composition frameworks are hindered by semantic fragmentation and a neglect of historical usage patterns. To address this problem, this research paper introduces a history-aware semantic framework that enhances the long-term robustness of service compositions by embedding APIs into a unified knowledge graph via a dedicated ontology that harmonizes heterogeneous data and services through semantic abstraction and domain-level alignment. Reliability of composition is further enhanced by incorporating historical composition data capturing co-occurrence and popularity. A genetic algorithm dynamically optimizes the weights of these metrics to select the Top-K most reliable mashup compositions. Extensive evaluation on the large-scale ProgrammableWeb benchmark (11,492 APIs and 7,336 mashups) demonstrates a statistically significant improvement of 26% in mashup reliability ($p<0.01$) compared to state-of-the-art baselines. While this work focuses on Intelligent Media mashups as a representative application domain, the proposed framework is applicable to general Web and IoT API composition. The integration of semantic abstraction, historical analysis and optimization provides a robust solution to generate resilient, adaptive and highly reliable service mashups, offering a useful approach to support the long-term adaptability and reuse of complex service ecosystems.

KEYWORDS

Intelligent media, Media mashups, Semantic integration, IoT and Web APIs.

1. INTRODUCTION

Modern software applications increasingly rely on the composition of diverse Web and IoT services in order to deliver rich and adaptive functionalities. At the same time, there is a growing need for service compositions that are not only functional, but also robust and reusable over time. In this context, the rapid expansion of the Web [1] and the Internet of Things (IoT) [2] has created unprecedented opportunities, producing massive volumes of interconnected data from APIs, sensors, devices and services. However, this dynamism introduces significant challenges-namely, semantic heterogeneity, interoperability breakdowns and an inability to generate reliably reusable mashups [3].

Mashups have emerged as a practical means of harnessing this rich ecosystem by enabling the creation of custom applications that combine functionalities from multiple Web and IoT APIs [4], allowing for new business opportunities [5]. For example, geospatial APIs can be merged with social-media streams to create context-aware cultural or tourism applications.

A fundamental challenge in developing such applications lies in the composition of mashups, in which diverse data and service streams must be integrated into meaningful and cohesive services. Composition is the key step that enables heterogeneous Web and IoT APIs to be integrated into value-added applications [6]. In this paper, we define a mashup composition as the process of selecting and combining a set of APIs to fulfill a specific functional requirement, with an emphasis on reliability and reusability of the resulting service bundle. Yet, the richness of available APIs-from sensors and social platforms to mapping and multimedia services-also carries drawbacks. One major drawback is interoperability, due to APIs exposing information in incompatible formats, it relies on different communication protocols, adopting varying structural designs [7]. This heterogeneity stems from the multitude of providers, each with its own way of developing, publishing and delivering resources, This leads to interoperability breakdowns and integration issues that hinder adaptive and reliable services [8].

1. M. A. Belfedhal is with EEDIS Research Lab., Djillali Liabes Uni. of Sidi Bel Abbes, Algeria. Email: mohammed.belfedhal@gmail.com
2. M. Malki is with LabRI Research Laboratory, Ecole Supérieure en Informatique de Sidi Bel Abbes, Algeria. Email: m.malki@esi-sba.dz
3. A. Khiat is with EEDIS Research Lab. and Dr. Moulay Tahar University of Saida, Algeria. Email: khiat.abderrahmane@univ-saida.dz

One solution to overcome this heterogeneity is to apply Linked Data principles [7] which extend traditional mashups by employing a unified data model known as the Resource Description Framework (RDF) model to lift API descriptions and then combine them leading to the emergence of Semantic Mashup.

A review of current semantic mashup and API composition frameworks reveals persistent limitations that reduce their efficiency for building reliable and reusable applications. These include: (1) semantic fragmentation, where a lack of well semantically structured RDF-based models prevents full interoperability; (2) ignoring historical usage patterns, such as co-occurrence and popularity, which prevents the application from learning from through compositions; (3) fixed weighting, where reliance on unchanging parameters limits adaptability to evolving contexts; and (4) functional gaps, where recommendation methods focus on clustering or complementarity, but overlook the functional variety needed for diverse services.

In this research, we introduce a history-aware semantic approach that overcomes these gaps and advances the state of reliable service composition. Our approach transforms heterogeneous Web and IoT APIs, together with their composition history, into semantically enriched, well-structured knowledge representations. More precisely, by using Linked Data principles, APIs are lifted into RDF through a dedicated API ontology and aligned with relevant domain ontologies selected according to user requirements. This semantic enrichment not only enables the seamless integration of disparate service resources, but also supports the discovery of more accurate and contextually relevant compositions through semantic interoperability. Besides semantic similarity, our framework enhances reliability further by incorporating historical composition data, such as Co-Occurrence-the frequency with which APIs have been combined - and Popularity - the extent of their reuse and evaluation within existing mashups. These factors are dynamically optimized through a genetic algorithm to compute the best-performing compositions. By unifying APIs into a shared semantic space and embedding history-aware reliability analysis, the framework delivers more robust, adaptive and reusable integrations, thereby contributing to the development of next-generation service-oriented applications and services.

Sustainability encompasses environmental, economic, social and technical dimensions. In this work, we address only one aspect: reusability - the ability of a mashup to remain robust and adaptable over time, reducing maintenance effort. We do not claim to address other dimensions, such as energy consumption. Additionally, we evaluate our general framework on Intelligent Media mashups as a challenging representative domain (involving video, audio, social and IoT APIs) without implying coverage of all mashup types.

To summarize, the main contributions of this research work consist of:

- Designing and developing a new API ontology and a JSON-to-RDF mapping mechanism for the semantic enrichment of heterogeneous, service-oriented APIs.
- Introducing a history-aware reliability model that integrates functional similarity, co-occurrence and popularity metrics within the RDF-based descriptions.
- Developing an adaptive optimization mechanism using a genetic algorithm to dynamically balance these metrics for recommending the Top-K reliable service mashups.

The paper is structured as follows: Section 2 reviews related work and clearly delineates the research gaps. Section 3 details our proposed framework. Section 4 presents the experimental evaluation and discusses the results. Finally, Section 5 concludes the paper and outlines future work.

2. RELATED WORK

Efforts to enhance automation in service integration and composition have inspired numerous approaches that are increasingly relevant to modern distributed systems. By enriching Web and IoT services with semantic descriptions, these methods aim to facilitate adaptive and context-aware applications that can integrate heterogeneous resources.

Early foundational work [9] provided ontologies and rule-based models that laid the foundation for semantically enriched services. WSMO-Lite [10] extended this by semantically annotating RESTful services, enabling applications to link APIs with contextual data. Similarly, scripting approaches, such as "S" [11], defined interaction resources for dynamic integration, while behavior modeling tools, like

ReLL [12] and metadata exposure methods, such as LRDD [13], supported richer interaction with Web APIs. RESTdesc [14] and RESTdoc [15] applied RDF-based templates to facilitate semantic discovery and the W3C HTTP vocabulary in RDF [16] helped bridge Web communication with semantic interoperability. More formal interaction models, such as pi-calculus [17] and triple space computing [18], define rigorous service interaction frameworks. In parallel, Linked Web API datasets [19]-[20] and semantic link discovery approaches [21] contributed to organizing and connecting API knowledge at scale. User-centered efforts, such as SAREST [22], SAWADL [23], introduced semantic annotation standards to enable discovery and generation of mashups, while visual tools [24] allowed non-experts to assemble services *via* drag-and-drop interfaces. Collectively, these approaches established important foundations by showing how semantic enrichment can transform fragmented APIs into interoperable and reusable services.

IoT-driven systems have adopted mashup and semantic composition techniques to deliver adaptive, user-centric services. For instance, Node-RED [25] provides a drag-and-drop environment that helps creators prototype IoT-enabled workflows. SSC [26] composes applications using live Linked Data and SPARQL, enabling dynamic linking of sensor data with external sources. Standards, such as OneM2M [27], use semantic reasoning to match heterogeneous sensor data, facilitating rich, real-time integrations. To improve interaction and accessibility, tools, like CoMMA [28], assist natural-language customization of mashups, while visual-composition systems automate the assembly of services. Additionally, the use of aspect-oriented programming (AOP) to modularize cross-cutting concerns in cloud-based systems, as demonstrated by the WeaveSim framework [29], illustrates how separating adaptive behaviors from core logic enhances scalability and reusability. On the front of content recommendation, neural embeddings and matrix factorization are used to suggest service compositions; user behavior pattern analysis refines recommendations toward more personalized delivery [30]-[31].

Systems, such as The Art of Video MashUp [32], combine semantic annotations and media features to help non-expert users craft creative mashups and A New Social Media Mashup Approach [33] merges social feeds and event metadata to detect and present content based on time, topic and location. Despite these advances, few systems jointly address semantic enrichment, adaptive delivery, composition history and reusability. Similarly, [34] enhances content recommendation accuracy by analyzing user-behavior patterns, while SSR [35] addresses redundancy through functional ranking, CSR [36] mitigates overlaps *via* category-aware clustering and SPR [37] optimizes complementarity using topic/tag analysis and Quadratic Knapsack models.

Despite their contributions, existing approaches reveal critical limitations when applied to building reliable and reusable systems. First, semantic fragmentation persists, which is observed in frameworks, such as Node-RED [25] and SA-REST [22]. Those frameworks lack well-structured RDF-based models to infer relationships between heterogeneous services, leading to fragmented integration. Second, historical blindness is common in works, such as the ML framework proposed by Xu et al. [31] and STRIM [38], leading to the neglect of patterns, such as co-occurrence and popularity, thus preventing systems from learning from past compositions and limiting adaptive optimization. Third, static weighting constrains adaptability, since context-aware systems (e.g., Boulakbech et al. [30]) rely on fixed parameters rather than dynamically adjusting reliability scores in response to evolving contexts. Finally, functional gaps or oversight limit personalization, as recommendation techniques, like CSR [36] and SPR [37], prioritize clustering or complementarity, but overlook nuanced functional diversity needed for adaptive and personalized services.

To overcome these gaps, we propose a history-aware semantic framework that integrates Linked Data principles (*via* RDF-based semantic lifting) with composition-history analysis (co-occurrence and popularity) and semantic functional analysis to identify reusable patterns across services. Reliability, metrics-semantic functional similarity, historical co-occurrence and popularity-are dynamically balanced using a genetic algorithm, ensuring contextually optimized Top-K compositions. The novelty of our approach can be understood from three dimensions. First, the dedicated API ontology was developed specifically for this framework. Second, while the genetic algorithm is an established technique, its application to per-request weight tuning for mashup-reliability scoring is a novel adaptation. Third, the API ontology enables semantic enrichment and interoperability, while the genetic algorithm dynamically adjusts the reliability weights; their integration into a single pipeline yields a robust Top-K mashup recommendation that neither component could deliver alone. In doing so, our approach delivers semantically enriched, adaptive and reusable integrations that advance the state of

service composition by enabling more reliable, context-aware and maintainable applications.

Table 1. Comparison of service composition frameworks.

Framework	Data Representation	Semantic Alignment & Matching	Historical Metrics	Optimization / Adaptivity	Reusability & Reliability
SA-REST [22]	XML + OWL annotations	Partial (ontology-based only)	Not considered	None (static)	Low - limited adaptability
Node-RED [25]	JSON flow-based	No formal semantics	Not considered	Manual selection	Low - manual integration
SSR [34]	RDF triples	Semantic category ranking	No historical data	Static weighting	Moderate-redundancy
CSR [35]	Category ontology	Cluster-based similarity	Popularity only	Fixed parameters	Moderate - no adaptive learning
SPR [36]	Topic graph	Shallow semantic grounding	Popularity + topic correlation	Static scoring	Moderate - no semantic matching
Our framework	RDF + Linked Data ontology	Full semantic alignment + ontology mapping	Popularity and co-occurrence in RDF	Dynamic weight optimization (GA)	High - adaptive, reliable, reusable

3. TOP-K MEDIA-INTELLIGENT MASHUP COMPOSITION *VIA* LINKED DATA AND HISTORICAL USAGE

In this section, we describe our new approach-the History-aware Semantic Mashup approach-which aims to enable reliable, reusable and adaptive composition of Web and IoT services. The framework addresses the limitations of existing mashup systems - such as semantic fragmentation, static weighting and lack of historical awareness-by integrating semantic enrichment, composition-history analysis and adaptive optimization (dynamic weights) in a unified model. Although we illustrate the approach using media-centric examples and evaluate it on media mashup datasets, the methodology is applicable to any domain requiring reliable API composition. The novelty of the proposed approach lies in three complementary aspects, each contributing to the overall reliability and reusability of mashup generation: (1) API Ontology and JSON-to-RDF Mapping: Unlike previous frameworks that rely on generic lightweight RDF transformation, our approach introduces a specialized ontology tailored for Web and IoT APIs. It enables JSON-to-RDF transformation, providing uniform semantic representation of heterogeneous services and improving interoperability across API ecosystems. (2) History-Aware Reliability Modeling: The framework captures historical composition knowledge by analyzing two key metrics-popularity (aggregated user ratings and satisfaction scores) and co-occurrence (joint appearance of APIs in successful compositions). These metrics are stored as RDF properties, allowing the system to infer reliability based on real usage trends rather than static design assumptions. (3) Adaptive Optimization *via* Genetic Algorithm: To dynamically balance semantic similarity and historical reliability, our proposed solution employs a genetic algorithm (GA) that optimizes the weighting of reliability factors. The GA iteratively evolves reliability scores through crossover and mutation operations, ensuring that the selected Top-K mashups achieve the best trade-off between functionality, popularity and co-occurrence for each specific composition request. By unifying these three dimensions-semantic enrichment, historical awareness and adaptive optimization - the proposed framework overcomes the rigidity of existing mashup approaches and establishes a robust foundation for reliable service integration.

As shown in Figure 1, our contribution lies in three dimensions: (1) Semantic enrichment: API descriptions are lifted into RDF using our API ontology and an Ontology-based JSON-to-RDF Mapping technique [39], enabling APIs to be treated as semantically linkable with other resources. Once lifted, they are aligned with domain ontologies according to user requirements, ensuring interoperability across heterogeneous streams. (2) History-aware reliability: We incorporate composition history by modeling popularity (how widely APIs or mashups have been reused), co-occurrence (how frequently APIs appear

together in compositions, reflecting their compatibility) and functional similarity (semantic proximity of their capabilities). These factors reflect the experiential knowledge of past mashups, analogous to how user feedback and usage data guide recommendations. (3) Adaptive optimization: A genetic algorithm dynamically assigns optimal weights to these measures, ensuring that Top-K reliable mashups are selected per user request.

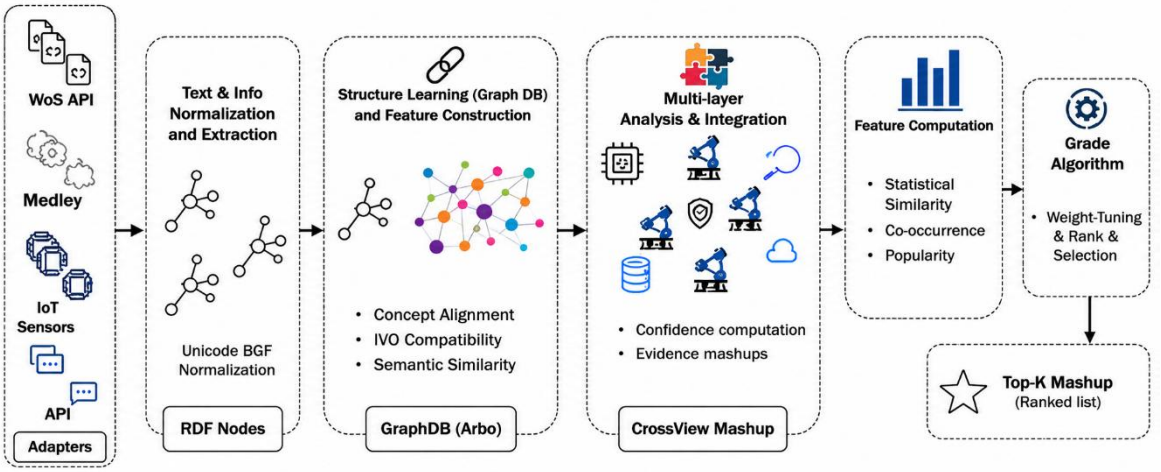


Figure 1. Selection of best API combinations.

We first present the overall architecture of our mashup framework; then, we highlight our main contributions.

3.1 Building Blocks of the Mashup Architecture

As shown in Figure 2, the framework is structured into five layers:

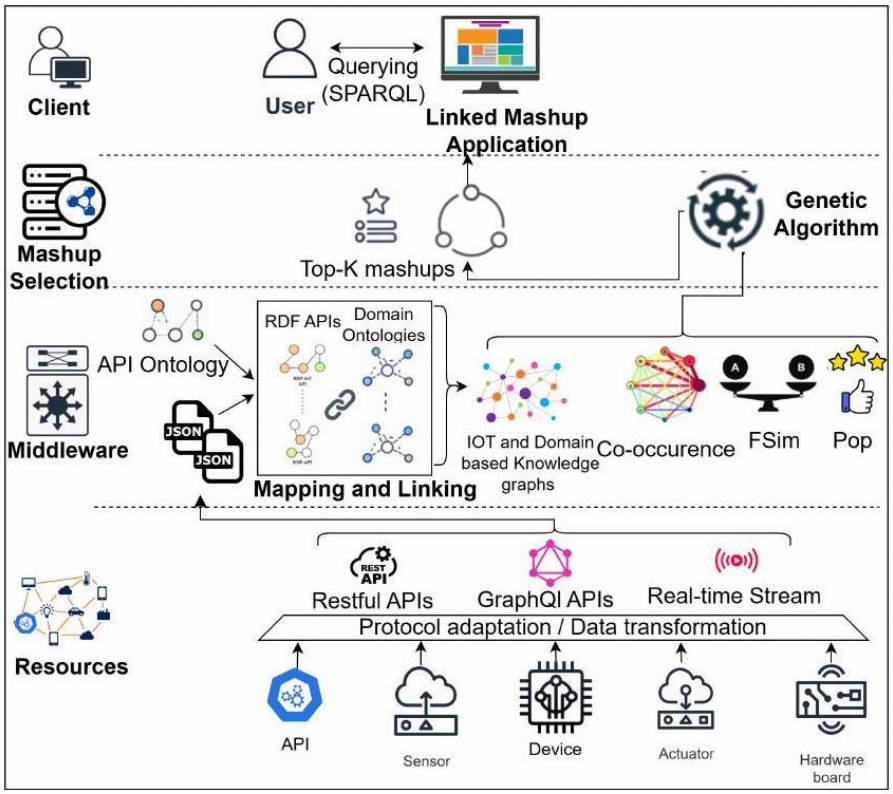


Figure 2. General architecture of the proposed mashup framework.

- Resources Layer: Includes Web APIs (e.g., mapping services), IoT devices (e.g., sensors) and other service interfaces, accessible *via* interfaces, such as GraphQL and REST. This layer supplies heterogeneous streams for mashups.

- Middleware Layer: Provides semantic interoperability by lifting raw API descriptions into RDF using our API ontology and aligning them with domain ontologies relevant to the user's request. This enhances semantic similarity and co-occurrence analysis for more accurate compositions.
- Mashup Selection Layer: Matches APIs to user requirements by evaluating similarity, co-occurrence and popularity, then applies a genetic algorithm to select the Top-K reliable compositions.
- Client Layer: Allows users to specify requirements and interact with the resulting mashups. Compositions are stored in a triple store and can be queried through a SPARQL endpoint for adaptive reuse.

Our main contributions revolve around the Middleware and Mashup Selection Layers: (1) semantic enrichment of APIs (Sub-ection 3.2) and (2) history-aware Top-K mashup composition (Sub-ection 3.3).

3.2 Lifting Web and IoT API and Mashup Descriptions into an RDF-based Model

We employed a semantic-transformation process to lift the raw data, mostly represented in JSON that describes the Web and IoT APIs and Mashups (see Figure 4 as an example) into an RDF-based representation (see Figure 5). This transformation aims at enhancing the interoperability and semantic richness of the description, making it more easily discoverable and linkable with other APIs on the Linked Data, particularly other Web and IoT APIs. Initially, we identified key components within the JSON structure, such as service name, description, functions, parameters and response properties.

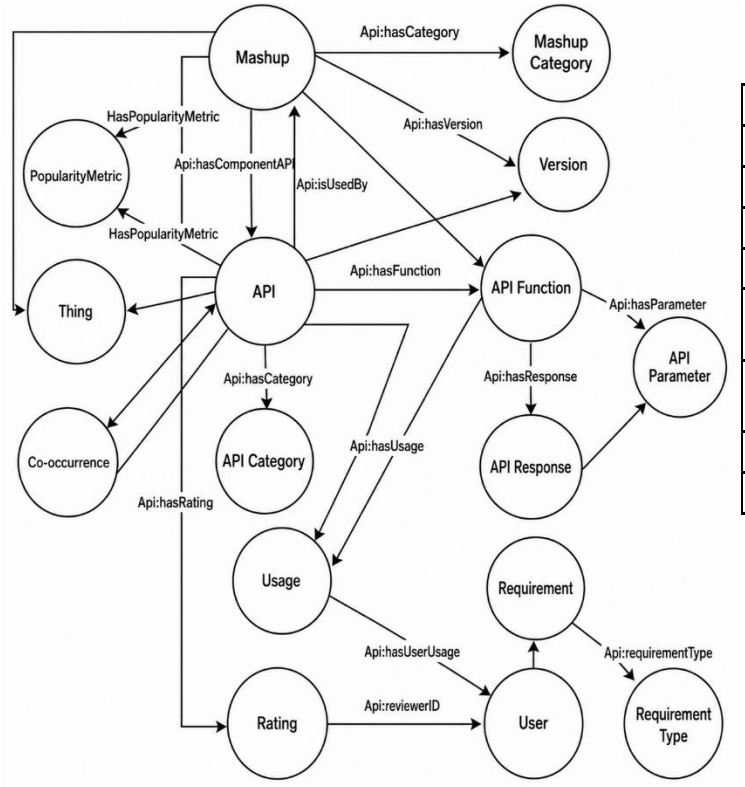


Figure 3. Proposed API-centric ontology.

Table 2. Ontology structural information measurement values.

Measurement	Value
Number of imported ontologies	2
Number of imported classes	15
Number of created classes	40
Subclass relations	17
Number of object properties	28
Number of data properties	20
Syntax language	English
Ontology engineering tool	Protégé

To do so, we developed the API-centric ontology by adopting the METHONTOLOGY ontology-development life cycle [40]. This methodology aligns with recent agile approaches that combine automated extraction with expert validation, as demonstrated in ontology development for collaborative innovation [41]. Table 2 provides measurement values of structural information of our developed ontology, while Figure 3 illustrates a portion of the ontology. The process had several stages: specification, conceptualization, formalization, implementation and maintenance. In the specification step, the goal, scope and intended users were defined, focusing on semantically representing and integrating API components and related domains for researchers and software engineers. Then, the conceptualization involved designing a conceptual model based on existing ontologies and iteratively developing the main concepts and relationships. Next, in the formalization step, the main concepts,

hierarchies and restriction axioms were represented, such as defining the API concept with its properties and relationships. Finally, the implementation is carried out using the Web Ontology Language (OWL) https://www.w3.org/OWL/ and Resource Description Framework (RDF) https://www.w3.org/RDF/, developed with Protégé and evaluated using SPARQL queries, while the maintenance ensures ongoing updates and refinements. Throughout the development of the ontology, key activities were conducted in parallel: knowledge acquisition through expert, domain-level evaluation with iterative feedback sessions and comprehensive documentation with labels and comments to describe the classes and properties. The ontology is evaluated against FAIR principles [42].

To ensure that the proposed ontology[1] is both semantically consistent and functionally adequate for supporting mashup composition, we performed a formal validation process using Protégé and its plugin-the Pellet reasoner. Consistency checking confirmed that no logical contradictions existed among class hierarchies, object properties, or data property domains and ranges, ensuring that ontology axioms were coherent. Beyond structural validation, a set of competency questions was defined to evaluate whether the ontology could effectively answer domain-relevant queries, such as: (1) Which APIs provide geolocation data usable in a travel mashup? (2) Which APIs can extract or analyze text features for content analysis? (3) Which combinations of APIs have been most frequently co-used in historical mashups? The ontology successfully resolved all competency questions using SPARQL queries over the generated RDF dataset, demonstrating its adequacy for semantic discovery and interoperability.

The connection from semantic lifting to recommendation is straightforward: user requirements are mapped to domain ontologies, which then query the RDF store for APIs the descriptions of which match these concepts. Candidate APIs are scored using three normalized metrics (range [0,1]) stored directly as RDF properties: resource:similarityValue (semantic proximity *via* LogMap), resource:coOccurrenceValue (joint usage frequency) and resource:popularityValue (aggregated user ratings). These scores feed into the reliability equation, the weight of which is optimized per request by the genetic algorithm.

For example, a user requesting a travel mashup triggers the selection of maps, weather and hotel APIs. with our API ontology in use, these APIs are lifted to RDF; domain ontologies (geolocation, weather, points of interest), then link these RDF descriptions into a unified semantic space. Multiple potential mashups-each combining one API per category-are formed. To select the best mashup, the functional similarity (computed *via* LogMap), co-occurrence and popularity (both derived from historical mashup data) of these candidates are fed into the genetic algorithm, which optimizes the six weights in the reliability equation and returns the Top-K most reliable compositions.

The resource: similarityValue property quantifies semantic proximity between services. This value is calculated using LogMap to generate ontology-based embeddings, with cosine similarity determining the normalized score in the range [0,1]. For instance, a speech-to-text API and a text-translation API might achieve a similarity value of 0.8, reflecting their complementary functions.

resource: coOccurrenceValue captures the historical frequency with which APIs appear together in successful mashups. This value is calculated as the normalized count of joint occurrences divided by the total usage frequency of both services. For example, a mapping API and a weather-service API that frequently co-appear in travel applications would receive a high co-occurrence value, indicating proven compatibility and integration feasibility.

resource: popularityValue measures service adoption and user satisfaction through aggregated ratings and usage statistics from multiple sources. This metric ranges from [0,1], where higher values indicate wider community adoption and positive user feedback. A highly-rated social-media API with extensive documentation and active developer communities would exhibit elevated popularity scores.

These three normalized values serve as critical inputs to the reliability computation in Equation 2, enabling the Genetic Algorithm to dynamically balance semantic alignment, historical compatibility and community validation during mashup optimization. By embedding these semantic-proximity measures directly into RDF descriptions, our framework enhances service discoverability and supports informed composition decisions for both developers and non-experts.

[1] Public GitHub repository: https://github.com/abderrahmane-khiat/SIMM/blob/main/Ontology/api_mashup_ontology.owl

After enriching media API descriptions using our API ontology, domain ontologies are selected based on user requirements to interlink these RDF descriptions. In the media domain, such ontologies include MPEG-7 for structural video descriptors, schema.org/VideoObject for metadata, the Audio Ontology for sound events, or FOAF for linking social-media entities. These ontologies define shared concepts and relationships that allow heterogeneous APIs-such as a video streaming API, a captioning API and a sentiment analysis API-to be linked into a coherent semantic space. For example, MPEG7:VideoSegment→hasAnnotation→ schema:Caption, or schema:VideoObject → hasFeature → schema:Sentiment. This linkage helps media-intelligent tasks, such as generating context-aware video summaries, synchronizing subtitles with visual events, or adapting content delivery based on user preferences. In this way, RDF-based enrichment combined with domain ontologies enables mashups to provide semantically robust, adaptive and personalized Intelligent Media experiences.

3.3 Top-K Reliable Mashup Composition with Popularity, Co-Occurrence and Functional Similarity

After linking different APIs, we leverage composition history through three primary functions: Co-Occ (Co-Occurrence), Pop (Popularity) and FSim (Functional Similarity). These functions are combined in our reliability formula and applied simultaneously to identify the most robust compositions to retain. We first define the three functions, followed by the overall reliability function.

```
{
  "name": "Google␣Maps␣Directions␣Service",
  "description": "Get␣driving,␣biking,␣or␣transit␣"
    "directions␣between␣locations.",
  "functions": [
    {
      "name": "getDirections",
      "parameters": [
        {
          "name": "origin",
          "type": "string",
          "description": "Starting␣location␣address␣"
            "or␣coordinates"
        },
        {
          "name": "destination",
          "type": "string",
          "description": "Ending␣location␣address␣"
            "or␣coordinates"
        },
        {
          "name": "travelMode",
          "type": "string",
          "description": "Optional:␣'driving',␣"
            "'bicycling',␣or␣'transit'"
        }
      ],
      "response": {
        "properties": {}
      }
    }
  ],
  "resources": "https://developers.google.com/maps/"
    "documentation/directions"
}
```

Figure 4. JSON description of a map API.

```
@prefix rdf:
      <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix api: <http://example.org/api/> .
@prefix ex: <http://example.org/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

ex:GoogleMapsDirectionsService a api:Service ;
    rdfs:label "Google␣Maps␣Directions␣Service" ;
    api:description "Get␣driving,␣biking,␣or␣transit␣"
        "directions␣between␣locations." ;
    api:hasFunction ex:getDirections ;
    api:resources <https://developers.google.com/maps/"
␣␣␣␣␣␣␣␣"documentation/directions> ;
    api:similarAPI [
        api:uri <https://example.com/similarAPI1> ;
        api:similarityValue "0.8"^^xsd:float
    ] ;
    api:similarAPI [
        api:uri <https://example.com/similarAPI2> ;
        api:similarityValue "0.85"^^xsd:float
    ] .

ex:getDirections a api:Function ;
    rdfs:label "getDirections" ;
    api:hasParameter ex:origin, ex:destination,
        ex:travelMode ;
    api:hasResponse ex:responseGetDirections .
```

Figure 5. RDF representation of a map API.

- Co-Occ(APIa, APIb): Defines a value that is related to the number of combinations between API a and API b. The value of Co-Occurrence is calculated by the following formula 1:

$$\text{Co} - \text{Occ}(\text{APIa}, \text{APIb}) = \frac{\text{number of compositions involving } API_a \text{ and } API_b}{\text{Total number of compositions involving } API_a \text{ or } API_b} \quad (1)$$

We did not limit ourselves to APIs, we also considered mashups that were already built. In order to distinguish between APIs and Mashups, we denoted MCo-Occ(M_a, M_b) for the Co-Occurrence calculation between two combinations of APIs.

- Pop(API_a) : Defines a value of the evaluations given by users to assess each API. To distinguish between APIs and Mashups, we denoted MPop (M_a) for the evaluations given by users to evaluate the combination of APIs. The value returned by Pop(API_a) or MPop(M_a) is a real number between 0 and 1.

- FSim(API_a, API_b) : The functional similarity used in our approach integrates the LogMap system to compute the similarity between Web and IoT APIs and compositions. LogMap [43]

is a scalable ontology-matching system with built-in reasoning and repair. It indexes input ontologies lexically and structurally, using inverted lexical indexes and external lexicons, like WordNet, to identify synonyms. It also leverages class hierarchies and employs reasoning to minimize inconsistencies in the aligned ontology. The LogMap system performs instance matching e.g., finding owl:sameAs property between linked data resources. In our research, these resources present a description of each API. LogMap shows very good results in terms of F-measure in the SPIMBENCH Benchmark series (https://oaei.ontologymatching.org/2015/im/index.html) which consists of a dataset with the purpose of determining when two RDF descriptions describe the same CreativeWork.

Similarly, we did not limit ourselves to APIs, we also considered mashups that were already built. To differentiate between APIs and Mashups, we denote $\mathrm{MFSim}(M_a, M_b)$ for the similarity computed between two combinations of APIs. We noted that the selected correspondences between APIs and compositions are added to the RDF-based API descriptor.

With multiple Web, IoT and other APIs available, applying these three functions generates four matrices (FSim, MFSim, Co-Occ, MCo-Occ) and two vectors (Pop, MPop). Together, they quantify similarity, historical co-use and user adoption, all of which are essential indicators of reliability in mashups.

To combine these indicators, we defined the Reliability Formula below. It calculates the composability between APIs and mashups, ensuring that only those compositions most likely to produce coherent and adaptive services are retained:

$$\begin{aligned}\mathrm{Reliability}(X) = {} & w_f \cdot \frac{1}{n}\sum_{i=1}^{n} \mathrm{FSim}_i + w_c \cdot \frac{1}{n}\sum_{i=1}^{n} \mathrm{CoOcc}_i + w_{fm} \cdot \frac{1}{n}\sum_{i=1}^{n} \mathrm{MFSim}_i + w_{cm} \\ & \cdot \frac{1}{n}\sum_{i=1}^{n} \mathrm{MCoOcc}_i + w_p \cdot \frac{1}{n}\sum_{i=1}^{n} \mathrm{Pop}_i + w_{pm} \cdot \frac{1}{n}\sum_{i=1}^{n} \mathrm{MPop}_i \end{aligned} \quad (2)$$

While each element denoted the APIs of a composition, $w_f, w_{fm}, w_c, w_{cm}, w_p$ and w_{pm} denoted the weights attributed respectively to the functional similarity, co-occurrence and popularity of both APIs and mashups.

The six weights are bounded between 0 and 1 and are initialized to 0.5 to give equal importance to all metrics before optimization.

The final step was to select the Top- K compositions identified by the Reliability Formula. Because the weights of FSim, Co-Occ and Pop directly determine the outcome, we employed the Genetic Algorithm (Algorithm 1) to optimize these weights. This ensures that the selected mashups are not only statistically reliable, but also contextually adaptive, producing compositions that align with user needs and semantics.

Candidate mashups are generated by, first, grouping APIs into functional categories (e.g., mapping, weather, social) based on their descriptions and the selected domain ontologies. For a user request that requires m distinct functionalities, we formed all combinations of one API from each of the m relevant categories. This yielded an initial candidate pool the size of which depends on the number of available APIs per category. From this pool, a random sub-set of $50 - 100$ candidate mashups was selected as input to the genetic algorithm to keep optimization computationally feasible. The sub-set is drawn uniformly without replacement, giving each possible combination an equal chance of being chosen. The random-number generators were not initialized with a fixed starting point. No explicit pre-filtering stage is applied; instead, the reliability scoring (Equation 2) naturally assigns low scores to low-quality combinations, so they are unlikely to appear among the Top-K recommendations. The size of the sub-set follows established guidelines for genetic-algorithm population sizes: De Jong [44] showed that $50 - 100$ individuals provide a good trade-off between diversity and convergence, a recommendation widely adopted in the GA literature. The value $K = 5$ was chosen, because it is the most commonly used setting in the API and mashup recommendation literature, offering a good balance between recall and precision. A systematic sensitivity analysis of K has not been performed.

Algorithm 1 outlines the weight-optimization process. The GA receives the randomly selected sub-set of candidate mashups, each already scored by the initial reliability function. For example, if a user

requests the creation of a travel planner, the system may combine mapping APIs (M1-M3), weather APIs (W1-W3) and hotel APIs (H1-H3) into multiple candidate mashups, such as M1-W2-H2 or M3-W3-H1. These candidates, ranked by the initial reliability function, form the pool from which the optimization sub-set is randomly sampled.

Algorithm 1. Genetic Algorithm weights optimization

Require: Subset of candidate mashups M_{sub}, Validation set V, Number of individuals N, Crossover rate C_{rate}, Mutation rate M_{rate}

Ensure: Optimized weight vector $\mathbf{w}^* = (w_f^*, w_c^*, w_p^*, w_{fm}^*, w_{cm}^*, w_{pm}^*)$

Initialize population P with N random weight vectors $\mathbf{w} = (w_f, w_c, w_p, w_{fm}, w_{cm}, w_{pm})$ where each $w_i \in [0,1]$

▷ Each individual encodes a possible weighting configuration for API- and mashup-level reliability

repeat

for each individual $\mathbf{w}$ in P **do**

Apply weights $\mathbf{w}$ to compute API-level reliability for each component API

▷ Reliability(API) $= w_f \cdot F\text{ Sim} + w_c \cdot CoOcc + w_p \cdot \text{Pop}$

Aggregate API reliabilities to compute mashup-level reliability

▷ Reliability(Mashup) $= w_{fm} \cdot F\text{ Sim}_m + w_{cm} \cdot CoOcc_m + w_{pm} \cdot \text{Pop}_m$

Rank mashups in M_{sub} by their computed reliability and select Top- K

▷ Identify the most promising mashups for validation

Evaluate selected mashups against validation set V

▷ Compute precision, recall, and F-measure relative to the validation ground truth

Compute fitness $(\mathbf{w})$ = F-measure $(\mathbf{w})$

▷ Assign the F-measure as the fitness value of the weight configuration

end for

Apply tournament selection and crossover (simulated binary) with probability C_{rate}

▷ Combine high-performing configurations to explore new solutions

Apply polynomial mutation on weight vectors with probability M_{rate}

▷ Introduce diversity by slightly altering weight values

Update population P with the new offspring

▷ Retain best-performing configurations for next generation

until F-measure improvement < 0.001 for 20 consecutive generations or maximum 100 iterations reached

▷ Stop when fitness plateaus or iteration limit is reached

return $\mathbf{w}^* = (w_f^*, w_c^*, w_p^*, w_{fm}^*, w_{cm}^*, w_{pm}^*)$

▷ Output the optimal weights for reliable and reusable mashup composition

The GA does not create new mashup compositions from scratch, but it instead refines the reliability-evaluation process by learning optimal weight configurations for the specific request at hand. At each generation, the GA:

(a) Applies the current weight configuration $\mathbf{w} = (w_f, w_c, w_p, w_{fm}, w_{cm}, w_{pm})$ (all weights in [0,1]) to compute API-level and mashup-level reliability scores for the sub-set of mashups;

(b) Performs tournament selection, simulated binary crossover (SBX) and polynomial mutation operations on the weight vectors (mutation rate $= 0.05$, crossover rate $= 0.8$);

(c) Evaluates the resulting reliability scores to compute precision, recall and F -measure as fitness metrics against a validation sub-set (20% of the candidate pool);

(d) Updates the population of weight configurations according to their fitness values until convergence criteria are met.

(e) The GA terminates when the F-measure improvement is less than 0.001 for 20 consecutive generations or a maximum of 100 generations is reached, whichever occurs first.

This iterative process establishes a dynamic balance between the semantic characteristics of APIs and their historical usage patterns: the initial candidate generation provides meaningful mashup structures based on semantic compatibility, while the GA continuously learns how to balance functional similarity, co-occurrence and popularity indicators across both API and mashup levels. The outcome is an adaptively weighted reliability model that enhances the robustness and reusability of mashup composition.

4. EXPERIMENTAL STUDY

This section details our experimental study.

4.1 Evaluation Dataset

To the best of our knowledge, there is currently no standardized benchmark for evaluating API integration and composition in service-oriented systems. Therefore, we curated an existing dataset - "Data Correction and Evolution Analysis of the ProgrammableWeb Service Ecosystem" [45] resource[2] by 1) filtering for APIs with categories relevant to common mashup domains (e.g., Mapping, Social, Weather, Media, IoT) based on keyword matching of ProgrammableWeb category tags 2) augmenting each API with a popularity metric obtained through systematic searches across multiple online API directories and developer forums (e.g., RapidAPI, API.io), providing a realistic assessment of API relevance and current usage trends. Popularity scores from different sources were normalized to the [0,1] range using Min-Max scaling; APIs with no available ratings were assigned a neutral value of 0.5. It was noted that the original dataset already includes co-occurrence of APIs. After cleaning and validation, the final curated dataset[3] contained 11,492 APIs and 7,336 mashups.

Next, to validate the performance of our system, we conducted a user test with 90 participants. Participants were asked to enter their requirements followed by a semi-structured interview. The goal of this study was to evaluate how our approach generates the best mashups that can answer different user requirements.

The dataset encompassed APIs that provide sensor data, social-media content, geospatial information, authentication and authorization services, blockchain interactions, business analytics and currency-exchange data.

4.2 Participants' Task and Evaluation Metrics

We recruited users by invitation to participate in our pilot study. The mailing list was formed by one author of this research (a student) by inviting his fellow students. In the end, we sent one round of invitations to 173 students to come up with 2 ideas of what they would like to have as a mashup system and we received 90 responses (52% response rate). Participants were aged 20-26 years, predominantly computer-science students with prior exposure to API-based systems. This sample size is adequate for an initial proof-of-concept evaluation; however, we acknowledge that the exclusive use of student participants may limit generalizability to professional developer communities. Accordingly, the findings should be regarded as a proof-of-concept rather than a definitive validation. A broader study with industry practitioners is planned for future work. It is worth mentioning that gathering free participants was very difficult and only 90 out of 173 accepted to participate. Furthermore, we did not pay participants on site or online *via* crowd platforms (e.g., MTurk) to avoid influencing the results of the evaluation.

For the task, we deployed our system on the web and we told participants that the higher their points were, the more they were promoting research. It was a form of motivation. Next, we asked them to use our system without any tutorial in order to get the first feedback about our system. The overall task (10 minutes) consisted of a 5-minute session for interacting with the system validation, followed by a 5 - minute interview. At the end, we gathered 180 systems.

Next, to validate our approach in generating mashups that best meet the user requirements, we built a ground truth by sharing the 180 systems among the participants and asking them to rate the best system (good, not specifically, not at all). The systems that received more than 10 good ratings were retained (this threshold was chosen to ensure that at least 10% of the 90 participants agreed on the quality of a system, providing a basic consensus filter). At the end, we ended up with 120 systems.

This ground truth created by participants allowed us to use measures such as precision (3), recall (4) and F-measure (5) to determine the new compositions' reliability. In the following, we define the measures Recall, Precision and F-measure used to evaluate the performance of our approach:

---

[2] Original dataset available at: https://github.com/HIT-ICES/Correted-ProgrammableWeb-dataset/tree/main/data

[3] Final dataset available at: https://github.com/abderrahmane-khiat/SIMM/tree/main/Media%20APIMashup%20dataset

- Precision: it denotes the precision level of selected resources (Web and IoT APIs), indicating the ratio of the chosen relevant APIs to the total number of chosen APIs. It is formally defined as follows:

$$\text{Precision } = \frac{R(A) \cap C(A)}{C(A)} \tag{3}$$

- Recall: it refers to the recall value of selected resources (Web and IoT APIs), indicating the ratio of the chosen relevant APIs to the total number of actual relevant APIs. It is formally defined as follows:

$$\text{Recall } = \frac{R(A) \cap C(A)}{R(A)} \tag{4}$$

- The F-measure: a tradeoff metric that balances recall and precision and is calculated using the formula:

$$F\text{-measure } = \frac{2 \times \text{ Precision} \times \text{Recall}}{\text{Precision } + \text{ Recall}} \tag{5}$$

Higher values of Recall, Precision and F-measure indicate that the system more accurately generates mashups that align with the user requirements.

All experiments were implemented in Python, running on a Windows 10 computer equipped with an Intel Core CPU at 2.8GHz, 16 GB RAM. The complete source code, including the framework implementation, baselines re-implementation and evaluation scripts, is publicly available for reproducibility under a GitHub repository[4].

4.3 Evaluation Protocol

For the evaluation of our approach, we conducted three studies:

- In the first study, we compared our approach with three systems: SSR (scores services individually, causing redundancy), CSR (groups services by function, but uses coarse cluster-level compatibility) and SPR (leverages topic/tag/history data, but assumes functional similarity ensures compatibility). Although these are API recommendation systems, their ability to compose compatible services provides a meaningful benchmark for evaluating mashup generation. To ensure a rigorous and fair comparison, we re-implemented SSR [35], CSR [36] and SPR [37] according to their original specifications. All baselines were evaluated under identical conditions using our curated dataset, with parameter settings aligned with their respective publications (e.g., Top-K values, similarity thresholds). This eliminates potential confounding factors when comparing performance metrics.

  In addition, we conducted tests of statistical significance of the comparative results using paired t-tests on the F-measure scores across each evaluation, with the null hypothesis stating no difference in performance between our approach and each baseline. We used the standard 95% confidence level and reported the resulting confidence intervals to quantify the precision and stability of the performance estimates.

- The second part of our investigation demonstrates the value of the metrics employed in our proposed solution by analyzing them both separately and in combination.
  - Functional Similarity (FSim): This baseline considers functional similarity between services based on their descriptions.
  - Popularity (POP): This method ranks services based on the number of times they have been used in mashups.
  - Co-Occurrence (CO): This baseline considers the frequency with which services appear together in mashups.

- In the third study, we conducted multiple experimental trials to evaluate how our GA algorithm performs against traditional methods in determining optimal values for the six parameters (w_f, $w_c, w_p, w_{fm}, w_{cm}, w_{pm}$), which correspond to the metrics Co-Occ, FSim, Pop, MFSim,

4 Code repository: https://github.com/abderrahmane-khiat/SIMM/tree/main/Code

MCoOcc and MPop in Equation 2. More specifically, we compared our GA algorithm with 1) static approaches that apply fixed weight values regardless of context in which all weights equal to $1/6$ and 2) empirically tuned static weights where those weights are determined through preliminary experiments, e.g., $w_f = 0.5, w_c = 0.4, w_p = 0.6, w_{fm} = 0.6, w_{cm} = 0.5, w_{pm} = 0.8$.

Our designed genetic algorithm was based on preliminary experiments and established GA literature. A population size of 100 was found to provide sufficient diversity. Similarly, the crossover rate (0.8), mutation rate (0.05) and termination criterion (20 stagnant generations) fall within commonly recommended ranges and demonstrated stable convergence across repeated runs. Additionally, we investigated the effect of varying the Top-K APIs and candidate mashups on the system's final recommendations, providing insights into how parameter tuning and selection impact the accuracy and adaptivity of mashup generation.

4.4 Results

4.4.1 Comparative Evaluation Results

We evaluated the resulting mashup compositions using our reliability formula against the ground truth. Table 3(a) presents the Precision, Recall and F-measure for our approach compared with the SSR 35, CSR [36] and SPR [37] systems. The results demonstrate that our framework achieves higher precision and recall across all metrics, indicating a significant improvement over baseline methods. Paired t-tests revealed that the improvements in F-measure were statistically significant with $p < 0.01$ for all three comparisons (*vs.* SSR: $t = 9.12, p < 0.0001$; *vs.* CSR: $t = 8.75, p < 0.0001$; *vs.* SPR: $t = 7.95, p = 0.0003$). SSR tends to produce redundant outputs by scoring services individually and CSR relies on coarse cluster-level compatibility, while SPR assumes that functional similarity alone guarantees compatibility.

Our approach systematically addresses these limitations by integrating three complementary metrics: Similarity, which captures the semantic relatability between heterogeneous APIs even when they use different specification terms; Co-Occurrence, which identifies frequently interacting APIs, reflecting compatibility, ease of integration and practical utility within a mashup; and Popularity, which reflects user-driven preferences and requirements. By combining these metrics, our system develops a more comprehensive understanding of the relationships among APIs and mashups, resulting in more accurate, adaptive and context-aware mashup recommendations. We achieved a higher recall than precision, which is common across all methods and is desirable for mashup recommendation. Our main goal was to find as many useful and compatible APIs as possible. In this context, missing a key API (false-negative) is more detrimental to mashup viability than suggesting extra, sub-optimal candidates (false-positives). Thus, our Recall-oriented performance profile aligns with the practical need for expansive discovery in adaptive ecosystems.

Table 3. Comparison results.

Metric	SPR	CSR	SSR	Our Approach
Precision	0.24	0.26	0.29	**0.37**
Recall	0.44	0.49	0.56	**0.68**
F-measure	0.31	0.34	0.38	**0.48**

(a) Performance comparison with baseline methods. Improvements are statistically significant ($p < 0.01$). Our approach achieves 26% improvement in F -measure.

FSim	Pop	Co-Occ	Combined
0.30	0.29	0.33	**0.37**
0.55	0.52	0.56	**0.68**
0.39	0.37	0.42	**0.48**

(b) Individual metric contributions *versus* combined performance

4.4.2 Parameter Combination Study

Table 3(b) illustrates the performance improvements achieved by combining the Similarity, Co-Occurrence and Popularity metrics compared to using each metric individually. These results show that each metric captures a distinct aspect of mashup quality. By simultaneously considering fine-grained compatibility, semantic relatability and user-driven relevance, our approach achieved superior performance.

We further investigated the effect of parameter application order on mashup selection, considering two strategies: sequential and parallel.

- Sequential approach: Parameters were applied in the following order Co-Occurrence → Similarity → Popularity, meaning that the selection of mashups was first influenced by API interaction patterns, then by semantic similarity and finally by user-oriented popularity.
- Parallel approach: All three metrics were applied simultaneously, weighted equally in the reliability formula and evaluated using Precision, Recall and F-measure.

The results indicate that the parallel approach provides more consistent and balanced outcomes, with precision and recall values generally 10% - 15% higher than the sequential approach. By treating all metrics simultaneously, the system avoids bias toward any single factor and produces more adaptive and accurate mashup recommendations.

4.4.3 Adaptive *vs.* Static Weight Optimization

This evaluation demonstrates the advantage of our genetic algorithm-based adaptive weighting approach compared to traditional static weighting methods. Unlike static approaches that apply fixed weight values regardless of context, our adaptive method dynamically computes optimal weight configurations for each individual mashup request through evolutionary optimization.

The key distinction lies in the optimization process: while static approaches use predetermined weights (typically equal weighting or values empirically tuned through preliminary experiments), our dynamic approach employs a genetic algorithm that, for every mashup request, explores the weight space, evaluating configurations against a validation sub-set to identify optimal balances between functional similarity, co-occurrence and popularity metrics specific to that request.

Table 4 presents the performance comparison between different weighting strategies that we used in our approach. The genetic-algorithm optimization consistently outperformed all static approaches, achieving superior F -measure (0.48) through balanced improvements in both precision (0.37) and recall (0.68).

Table 4. Performance comparison: Static weighting *vs.* Per-request genetic algorithm optimization.

Weighting Strategy applied on Our Approach	Precision	Recall	F-measure
Equal weighting (all weights = $1/6$)	0.31	0.59	0.41
Empirically tuned static weights	0.33	0.60	0.43
Genetic algorithm optimization	$\mathbf{0.37}$	$\mathbf{0.68}$	$\mathbf{0.48}$

The comparative evaluation revealed two findings:

- The optimal weight configuration varied significantly across different mashup requests, even within the same application domain. For instance, requests for travel planning mashups showed weights of $w_f \approx 0.85, w_c \approx 0.45, w_p \approx 0.90$, while social-media aggregation requests converged to $w_f \approx 0.75, w_c \approx 0.60, w_p \approx 0.95$. This is because the importance of each similarity, popularity or co-occurrence is different for each request. This fine-grained adaptation is only possible using genetic-algorithm optimization.
- Static approaches failed to capture these request-specific variations, leading to sub-optimal recommendations. Our adaptive method demonstrated over 10% better performance for novel or complex requests where static approaches make little use of historical patterns.

The results show that the optimal configurations of our genetic algorithm were highly request-dependent

with weights adjusting based on:

- The specificity and novelty of user requirements;
- The availability and characteristics of relevant APIs;
- The density of historical usage patterns in the target domain;
- The balance between semantic compatibility and practical integration concerns.

Regarding computational scalability, the GA optimization completed within 2-3 seconds per request on our test hardware. For larger API ecosystems, candidate generation can be accelerated using indexing structures to pre-filter APIs before semantic matching, ensuring that the approach remains practical for real-time applications.

5. CONCLUSION

In this research paper, we presented a novel history-aware semantic framework for generating reliable and reusable mashups that addresses key challenges in heterogeneous API integration and composition. To achieve this, we first developed an API ontology and JSON-to-RDF mapping for semantic enrichment of APIs to harmonize APIs and compositions for better API discovery. Second, we incorporated three complementary metrics-functional similarity, co-occurrence and popularity-within RDF-based descriptions in order to learn the historical composition patterns. Finally, we employed a genetic algorithm that performs dynamic weight optimization to produce top- k reliable mashup recommendations.

Experimental evaluation on a curated dataset of 11,492 APIs and 7,336 mashups demonstrated statistically significant improvements of 26% in F-measure over state-of-the-art baselines (SSR, CSR, SPR), with all comparisons achieving $p < 0.01$.

The results confirm that by unifying semantic interoperability with adaptive, context-aware reliability analysis, our approach promotes the long-term robustness and reusability of mashup compositions. We emphasize that our contribution to sustainability is limited to enhancing reusability-a critical, but narrow, dimension of the broader sustainability concept. The use of Intelligent Media mashups in our experiments serves as a representative case study; the framework itself is domain-agnostic and can be applied to general IoT and Web API composition scenarios. Generalizability to real-world API ecosystems remains to be established. Furthermore, in future work, we plan to address some limitations, such as conducting a systematic sensitivity analysis of K and initializing the random-number generators with a fixed starting point to ensure exact reproducibility, in addition to exploring more promising directions through machine-learning techniques enhanced by ontological knowledge leveraging structured semantics to further improve the accuracy, reliability and reusability of service compositions across application domains.

REFERENCES

[1] R. E. Roth, R. G. Donohue, C. M. Sack, T. R. Wallace and T. M. Buckingham, "A Process for Keeping Pace with Evolving Web Mapping Technologies," Cartographic Perspectives, no. 78, pp. 2552, 2014.

[2] Y. Qin et al., "When Things Matter: A Survey on Data-centric Internet of Things," Journal of Network and Computer Applications, vol. 64, pp. 137-153, 2016.

[3] E. Borgia, "The Internet of Things Vision: Key Features, Applications and Open Issues," Computer Communications, vol. 54, pp. 1-31, 2014.

[4] Y. Maaravi and B. Heller, "Digital Innovation in Times of Crisis: How Mashups Improve Quality of Education," Sustainability, vol. 13, no. 13, p. 7082, 2021.

[5] B. Beemer and D. Gregg, "Mashups: A Literature Review and Classification Framework," Future Internet, vol. 1, no. 1, pp. 59-87, 2009.

[6] F. Daniel and M. Matera, Mashups-Concepts, Models and Architectures, E-ISSN: 2197-974X, Springer Berlin, Heidelberg, 2014.

[7] A. K. Kalou and D. A. Koutsomitropoulos, "Towards Semantic Mashups: Tools, Methodologies and State of the Art," International Journal of Information Retrieval Research, vol. 5, no. 2, pp. 1-25, 2015.

[8] W. Pan, X. Xu, H. Ming and C. K. Chang, "Clustering Mashups by Integrating Structural and Semantic Similarities Using Fuzzy ahp," Int. J. of Web Services Research, vol. 18, no. 1, pp. 34-57, 2021.

[9] S. Handschuh, "Semantic Annotation of Resources in the Semantic Web," Chapter in Book: Semantic Web Services: Concepts, Technologies and Applications, Semantic Annotation of Resources in the Semantic Web, pp. 135-155, DOI: 10.1007/3-540-70894-4_5, 2007.

[10] P. Rodriguez-Mier et al., "An Integrated Semantic Web Service Discovery and Composition Framework," IEEE Trans. on Services Computing, vol. 9, no. 4, pp. 537-550, 2015.
[11] D. Bonetta, A. Peternier, C. Pautasso and W. Binder, "S: A Scripting Language for High-Performance Restful Web Services," ACM Sigplan Notices, vol. 47, no. 8, pp. 97-106, 2012.
[12] R. Alarcon, E. Wilde and J. Bellido, "Hypermedia-driven Restful Service Composition," Proc. of ICSOC 2010 Workshops, LNCS 6568, pp. 111–120, Springer, 2011.
[13] D. Cooper et al., Internet Engineering Task Force, Fremont, USA, 2008.
[14] R. Verborgh et al., "Description and Interaction of Restful Services for Automatic Discovery and Execution," Proc. of the Int. Workshop on Advanced Future Multimedia Services (AFMS), pp. 1-10, [Online], available: https://backoffice.biblio.ugent.be/download/2003291/2003308, 2011.
[15] D. John and M. Rajasree, "Restdoc: Describe, Discover and Compose Restful Semantic Web Services Using Annotated Documentations," Int. J. of Web & Semantic Technology, vol. 4, no. 1, p. 37, 2013.
[16] J. Koch, C. Velasco and P. Ackermann, "HTTP Vocabulary in RDF 1.0," W3C Working Draft, World Wide Web Consortium, 2011.
[17] R. Milner, "The Polyadic π-calculus: A Tutorial," Proc. of Logic and Algebra of Specification, in Book: NATO ASI Series, vol. 94, pp. 203-246, Springer, 1993.
[18] A. G. Hernández and M. N. M. García, "A Formal Definition of Restful Semantic Web Services," Proc. of the 1st Int. Workshop on Restful Design, pp. 39-45, DOI: 10.1145/1798354.179838, 2010.
[19] M. Dojchinovski and T. Vitvar, "Linked Web APIs Dataset," Semantic Web, vol. 9, no. 4, pp. 381-391, 2018.
[20] S. Omelkova and P. Küngas, "A Linked Data Model for Web API-s," Proc. of the 14th Int. Conf. in Business Informatics Research, pp. 48-63, DOI: 10.1007/978-3-319-21915-8_4, Springer, Estonia, 2015.
[21] F. Ardjani, D. Bouchiha and M. Malki, "An Approach for Discovering and Maintaining Links in RDF Linked Data," Int. Journal of Modern Education and Computer Science, vol. 9, no. 3, p. 56, 2017.
[22] J. Lathem et al., "SA-REST and (S)mashups: Adding Semantics to RESTful Services," Proc. of the IEEE Int. Conf. on Semantic Computing (ICSC 2007), pp. 469-476, Irvine, CA, USA, 2007.
[23] A. Malki, S. M. Benslimane, "Building Semantic Mashup," Proc. of ICWIT, pp. 40-49, [Online], Available: https://ceur-ws.org/Vol-867/Paper5.pdf, 2012.
[24] A. H. Ngu, M. P. Carlson, Q. Z. Sheng and H.-y. Paik, "Semantic-based Mashup of Composite Applications," IEEE Trans. on Services Computing, vol. 3, no. 1, pp. 2-15, 2010.
[25] N. Heath, "How IBM's Node-RED is Hacking Together the Internet of Things," TechRepublic, vol. 14, 2014.
[26] E. Prudhommeaux, "SPARQL Query Language for RDF," W3C Recommendation, [Online], Available: http://www.w3.org/TR/rdf-sparql-query/, 2008.
[27] A. Khattab et al., "Enhancing User Experience in IoT Mashup Using Semantic Technology," Proc. of the 2018 IEEE Global Conf. on Internet of Things (GCIoT), pp. 1-6, Alexandria, Egypt, 2018.
[28] S. Kim and I.-Y. Ko, "A Conversational Approach for Modifying Service Mashups in IoT Environments," Proc. of the 2022 CHI Conf. on Human Factors in Computing Systems (CHI '22), Article no. 345, pp. 1-16, DOI: 10.1145/3491102.351765, 2022.
[29] A. M. R. AlSobeh, S. AlShattnawi, A. Jarrah and M. M. Hammad, "WeaveSim: A Scalable and Reusable Cloud Simulation Framework Leveraging Aspect-oriented Programming," Jordanian Journal of Computers and Information Technology (JJCIT), vol. 6, no. 2, pp. 182-201, 2020.
[30] M. Boulakbech, N. Messai, Y. Sam and T. Devogele, "Configuration Approach for Personalized Travel Mashup," Concurrency and Computation: Practice and Experience, vol. 35, no. 11, 2023.
[31] Y. Xu, Z. Wang, H. Gao, Z. Jiang, Y. Yin and R. Li, "Towards Machine-learning-driven Effective Mashup Recommendations from Big Data in Mobile Networks and the Internet-of-Things," Digital Communications and Networks, vol. 9, no. 1, pp. 138-145, 2023.
[32] D. Cardillo et al., "The Art of Video Mashup: Supporting Creative Users with an Innovative and Smart Application," Multimedia Tools and Applications, vol. 53, no. 1, pp. 1-23, 2011.
[33] A. Troudi, C. A. Zayani, S. Jamoussi and I. Amous, "A New Social Media Mashup Approach," Proc. of Intelligent Systems Design and Applications Conf. (ISDA2016), Part of the Book Series: Advances in Intelligent Systems and Computing, vol. 557, pp. 677-686, 2017.
[34] L. Zhang et al., "JMF-SS: Joint Matrix Factorization for Web API Recommendation with Mashup-Mashup Similarities and API-API Similarities," Proc. of the 2024 27th IEEE Int. Conf. on Computer Supported Cooperative Work in Design (CSCWD), pp. 1219-1224, Tianjin, China, 2024.
[35] W. Gao and J. Wu, "A Novel Framework for Service Set Recommendation in Mashup Creation," Proc. of the 2017 IEEE Int. Conf. on Web Services (ICWS), pp. 65-72, Honolulu, HI, USA, 2017.
[36] B. Xia et al., "Category-aware API Clustering and Distributed Recommendation for Automatic Mashup Creation," IEEE Trans. on Services Computing, vol. 8, no. 5, pp. 674-687, 2014.
[37] J. Cao, Y. Lu and N. Zhu, "Service Package Recommendation for Mashup Development Based on a Multi-level Relational Network," Proc. of the 14th Int. Conf. on Service-Oriented Computing (ICSOC 2016), pp. 666-674, Banff, AB, Canada, October 10-13, 2016.

[38] A. Khiat, M. Benaissa and M. A. Belfedhal, "STRIM Results for OAEI 2015 Instance Matching Evaluation," Proc. of the 10th Int. Workshop on Ontology Matching OM-15 Collocated with the 14th Int. Semantic Web Conference (ISWC-2015), pp. 208-215, Bethlehem, USA, 2015.

[39] S. J. R. Méndez et al., "J2RM: An Ontology-based JSON-to-RDF Mapping Tool," ISWC (Demos/Industry), pp. 368-373, [Online], Available: https://ceur-ws.org/Vol-2721/paper593.pdf, 2020.

[40] M. Fernández-López, A. Gómez-Pérez and N. Juristo, "Methontology: From Ontological Art towards Ontological Engineering," Proc. of the Ontological Engineering (AAAI-97), [Online], Available: https://cdn.aaai.org/Symposia/Spring/1997/SS-97-06/SS97-06-005.pdf, USA, 1997.

[41] F. Kharbat, A. Alshawabkeh and M. Sharairi, "A Research-based Ontology for Collaborative Innovation: A Methodology Leveraging AI and Domain Expert Knowledge," Jordanian Journal of Computers and Information Technology (JJCIT), vol. 10, no. 03, pp. 265-280, 2024.

[42] D. Garijo and M. Poveda-Villalón, "Best Practices for Implementing FAIR Vocabularies and Ontologies on the Web," Chapter in Book: Applications and Practices in Ontology Design, Extraction and Reasoning, pp. 39-54, DOI: 10.3233/SSW200034, IOS Press, 2020.

[43] E. Jiménez-Ruiz and B. Cuenca Grau, "LogMap: Logic-based and Scalable Ontology Matching," Porc. of the10th Int. Semantic Web Conf. (ISWC2011), Part of the book series: Lecture Notes in Computer Science, vol. 7031, pp. 273-288, Germany, 2011.

[44] K. A. De Jong, An Analysis of the Behavior of a Class of Genetic Adaptive Systems, Thesis, University of Michigan, USA, 1975.

[45] M. Liu, Z. Tu, Y. Zhu, X. Xu, Z. Wang and Q. Z. Sheng, "Data Correction and Evolution Analysis of the Programmable Web Service Ecosystem," Journal of Systems and Software, vol. 182, p. 111066, 2021.

ملخص البحث:

يشكّل العدد المتزايد من واجهات برمجة تطبيقات الويب وإنترنت الأشياء غير المتجانسة تحدّياً جوهرياً لموثوقية التّطبيقات الحديثة وإمكانية إعادة استخدامها، حيث تعاني أُطُر تركيب واجهات برمجة التّطبيقات الحالية من التّجزئة الدّلالية وإهمال أنماط الاستخدامات السابقة. ولمعالجة هذه المشكلة، نقدّم إطاراً دلالياً مراعياً للتّاريخ، يعزّز متانة تركيبات الخدمات على المدى الطّويل، من خلال دمْج واجهات برمجة التّطبيقات في مخطط بياني معرفي موحّد عبر أنطولوجيا مخصّصة تنسّق البيانات والخدمات غير المتجانسة، من خلال التّجريد الدّلالي ومواءمة مستوى المجال. كذلك يتمّ تعزيز موثوقية التّركيب من خلال دمج بيانات التّركيب التّاريخية الّتي تسجّل التّواجد المشترك والشّعبية. حيث تعمل خوارزمية جينية على تحسين أوزان هذه المقاييس ديناميكياً لاختيار أفضل تركيبات الدّمج الأكثر موثوقية.

ويُظهر تقييم شامل على معيارٍ واسع النّطاق (11492 واجهة برمجة تطبيقات و 7336 عملية دمْج) تحسُّناً ذا دلالة إحصائية بنسبة 26% في موثوقية التّطبيقات المركبة مقارنة بالنماذج المرجعية. ومع أنّ هذا العمل يركّز على عمليات دمْج الوسائط الذّكية كمجال تطبيقٍ نموذجي، فإنّ الإطار المقترح قابل للتّطبيق على تركيب واجهات برمجة تطبيقات الويب وإنترنت الأشياء بشكلٍ عامّ. ويوفّر دمج التّجريد الدّلالي وتحليل التّركيبات السابقة والتّحسين حلّاً فعّالاً لإنشاء تطبيقات خدماتٍ مرنة وقابلة للتّكيُّف وذات موثوقية عالية، ممّا يقدّم منهجاً مفيداً لدعم قابلية التّكيُّف وإعادة استخدام أنظمة الخدمات المعقّدة على المدى الطّويل.

This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).