Reusing Semi-Specified Behavior Models in Systems Analysis and Design: The Web-Based Accelerated Search Case Study

Iris Reinhartz-Berger
University of Haifa
Carmel Mountain, Haifa 31905
iris@mis.haifa.ac.il

Dov Dori
Technion – Israel Institute of Technology
Technion City, Haifa 32000
dori@ie.technion.ac.il

Shmuel Katz
Technion – Israel Institute of Technology
Technion City, Haifa 32000
katz@cs.technion.ac.il

To demonstrate the OPM-based weaving process, we present here a case study of a Web-based accelerated search system. The system implements an algorithm for improving the performance of a Web search engine, which employs time-consuming search algorithms. The design of the Accelerated Search System includes two modules – the generic **Acceleration** module and the target **Multi Search** one. The **Acceleration** module [1] specifies a generic algorithm that reduces the execution time of an input-output part of a system by trying first to retrieve the output, which is determined by the input, from a local database. We assume that the sought Web-based items rarely change, so they are relatively static. This implies that results of subsequent activations of a query with the same input remain valid and can therefore be stored and retrieved to avoid executing the costly calculation each time a query with that input is submitted. If the entry is not already in the database, the algorithm activates a process that calculates or otherwise obtains the sought output and records it in the database to accelerate future executions of the query with the same input. The **Multi Search** module implements a new search engine that benefits from existing search engines by combining their results and ordering them according to a weighted score. In what follows we describe the three stages of the OPM-based weaving process for the Web-based Accelerated Search System.

1. Designing the Acceleration and Multi Search Modules

Figure 1 shows the OPM **Acceleration** module. At the top level system diagram shown in Figure 1(a), **Accelerating** requires **Input** to produce **Output** and affects (updates) the database **DB**. According to Figure 1(b), the **Accelerating** process zooms into (consists of) the subprocesses **DB Searching**, which searches the DB for an input-output entry, **Output Retrieving**, which retrieves the output if it was found in the database, and **Full Process Activating**, which activates the full-blown process of computing the output when necessary. As Figure 1(c) shows, during **Full Process Activating**, **Output Computing** first computes the output, and then **DB Updating** records the input-output pair in the database for future searches.

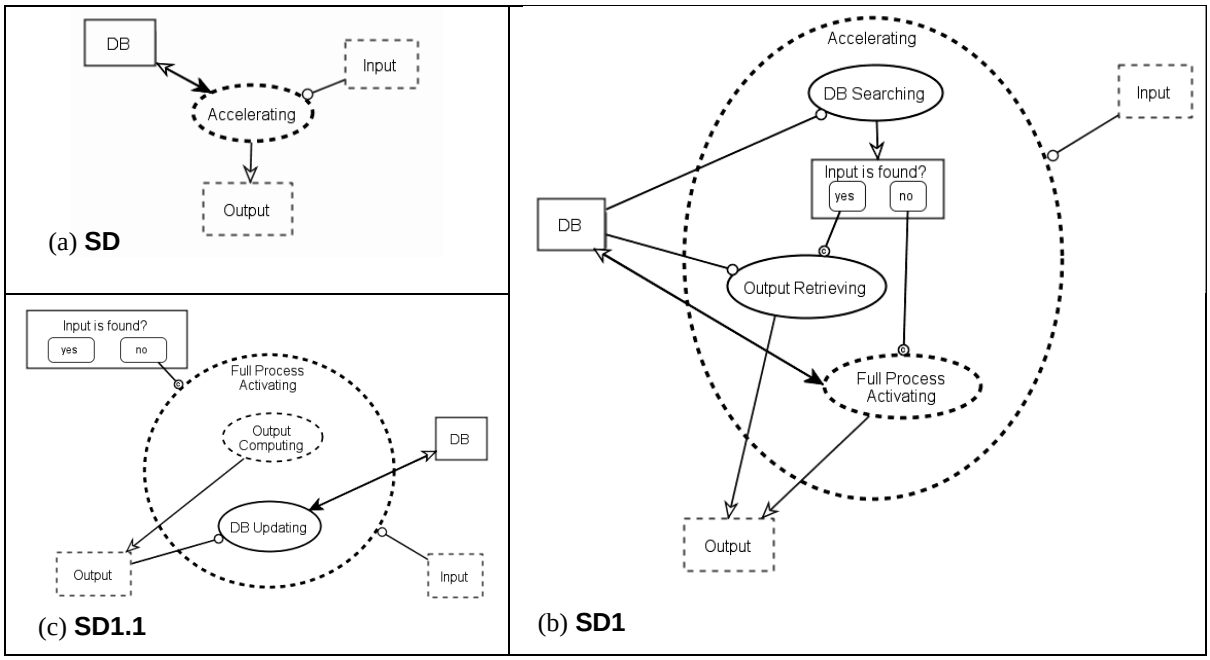

Figure 1. The **Acceleration** module. (a) **SD** is the top-level diagram. (b) **SD1** has **Accelerating** of **SD** in-zoomed. (c) **SD1.1** has **Full Process Activating** of **SD1** in-zoomed.

The **Acceleration** module contains two environmental objects, **Input** and **Output**, and one simple (atomic) environmental process, **Output Computing**. The refinement rule implies that **Full Process Activating** must also be environmental, since it contains an environmental

process. Following the same line of reasoning, **Accelerating**, which is at a yet more abstract level, must also be environmental. All the other things, including the object **DB**, the Boolean object "**Input is found?**," and the processes **DB Searching** and **Output Retrieving**, are systemic. They are internal to the **Acceleration** module and would not change when this module is woven into a target module. The OPM model of the **Acceleration** module synthesizes two possible scenarios (the input-output pair was found in the database and the input-output pair was not found in the database) into one behavior, enabling the reuse of complete behaviors rather than just structures or individual scenarios.

Figure 2 is an OPM specification of the **Multi Search** algorithm, executed by the **Multi Searching** process. Figure 2(a) is SD, the top-level diagram, which specifies the inputs, **Term** and **Query Result Message**, and outputs, **Query Message** and **Search Result**, of the **Multi Search** algorithm. Assuming that the algorithm operates concurrently on three different search engines, the diagram SD1 in Figure 2(b) shows that **Search Starting** requires **Term**, from which it creates three queries, one for each engine. **Query Sending** then creates from the three queries three **Query Messages**, which are sent in parallel to the relevant search engines (the sending part is not included in this model). **Result Collecting** waits until all three replies, called **Query Result Message 1, 2,** and **3**, arrive. It then outputs them as **Result 1, 2,** and **3**, respectively. Finally, **Result Merging** combines these three results to get the **Search Result**.

The **Multi Searching** process depends on the speed of each search engine, the network response time, and the number and size of results supplied by each search engine. Any combination of these factors can slow down the **Multi Searching** response time. To improve the system's performance, we weave the **Acceleration** module (Figure 1) into the **Multi Search** module (Figure 2), in order for the most recently searched terms and their corresponding

results to be saved in a local database. For each new query, this local database is searched before invoking the entire **Multi Searching** process, and only if the result is not found there, the system will execute the **Multi Searching** process.

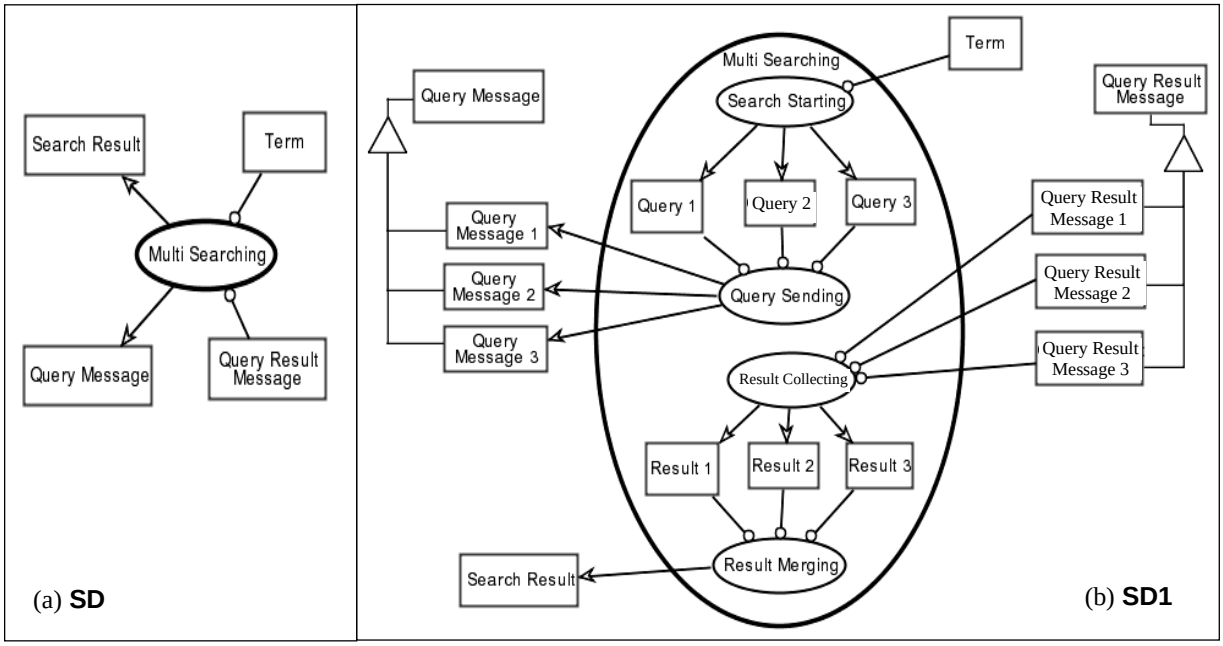

Figure 2. The **Multi Search** module. (a) **SD** is the top-level diagram. (b) **SD1** has **Multi Searching** of **SD** in-zoomed.

2. Creating the Accelerated Multi Search Module

Figure 3 shows the generic **Acceleration** module, derived from **SD1.1** of **Acceleration** in Figure 1(c), woven into **SD** of the **Multi Search** module in Figure 2(a) to create the woven module, called **Accelerated Multi Search**.

Three generalization-specialization relations connect the things in the generic module to the corresponding things in the target module. Two of these relations are between object classes, specifying that **Term** is an **Input** and that **Search Result** is an **Output**. The third generalization-specialization relation is between two process classes, specifying that the systemic **Multi Searching** process specializes the environmental **Output Computing** process.

In each of the three bindings, an atomic thing in the generic module generalizes a corresponding thing in the target module: **Input**, **Output,** and **Output Computing** generalize **Term**, **Search Result**, and **Multi Searching**, respectively. According to the intra-weaving rules, the environmental process **Full Process Activating** is implicitly bound to a default systemic process, called **Concrete Full Process Activating**, which includes just **Multi Searching**.

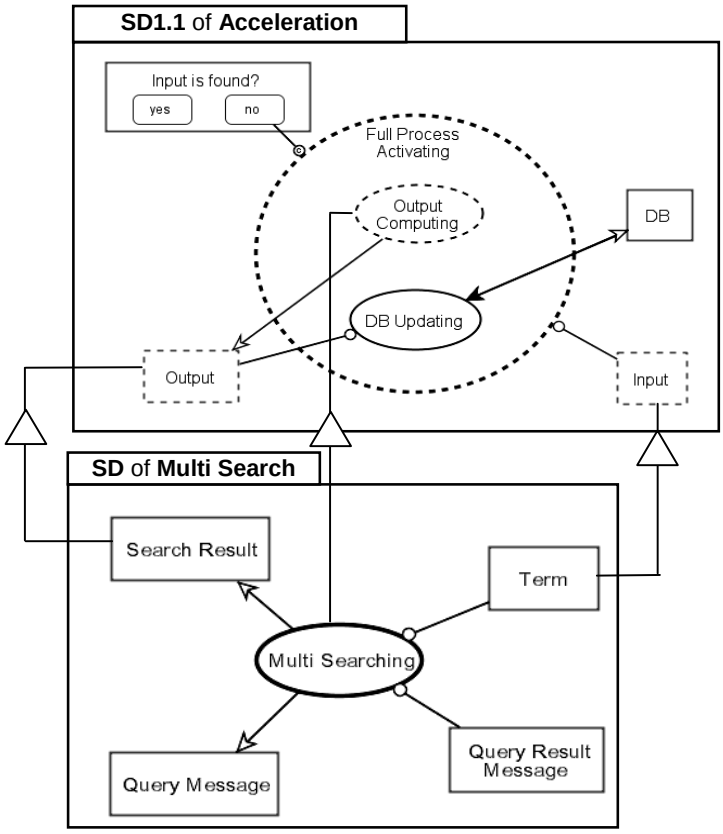

Figure 3. The **Accelerated Multi Search** module.

Merging the **Acceleration** and **Multi Search** modules would result in a complex explicit model equivalent to the woven module in Figure 3. However, the explicit model is both specific to the problem and harder to understand than the woven module. The generic nature of the **Acceleration** module in Figure 1 makes the same core architecture reusable for a variety

of related functions. Enhancements to the (non-merged) generic **Acceleration** module (such as periodic checks whether the local database values are still fresh) will automatically be reflected in any model into which this module is woven. Physically, the **Acceleration** module may reside in any repository. If continuous update of the modules is desired, new versions of the **Acceleration** module can be broadcast, published, or pushed to its customer applications whenever it is updated.

3. Refining the Woven Accelerated Multi Search Module

The equivalent semantics of the woven and the merged modules enables treating woven modules as either generic or target OPM modules in other weaving operations. The system architect can continue specifying the system into which a module has been woven as a complete application. We demonstrate two refinements that enhance the woven **Accelerated Multi Search** module obtained in Section 5.2. The first refinement improves the **Result Merging** algorithm within **Multi Searching** by treating **DB** as an additional input, while the second one adds to the system an entire generic **Log Recording** module. Any combination of these two refinements can be part of the system design.

Improving the **Result Merging** process is achieved by adding the capability to retrieve related term-result pairs from the database and use them to decide how to score the new search results. To carry out this function, we add in Figure 4 to the **Accelerated Multi Search** module an extra instrument link from **DB** (of the generic **Acceleration** module) to **Result Merging** (of the target **Multi Search** module). This makes **DB** an additional input to **Result Merging**. To be able to express this binding, the **Multi Searching** process is shown in **SD1** of the **Multi Search** module (see Figure 2(b)). Further zooming into **Result Merging** would

contain details specifying how the database (**DB**), added by the **Acceleration** module, improves the merging of the query results.

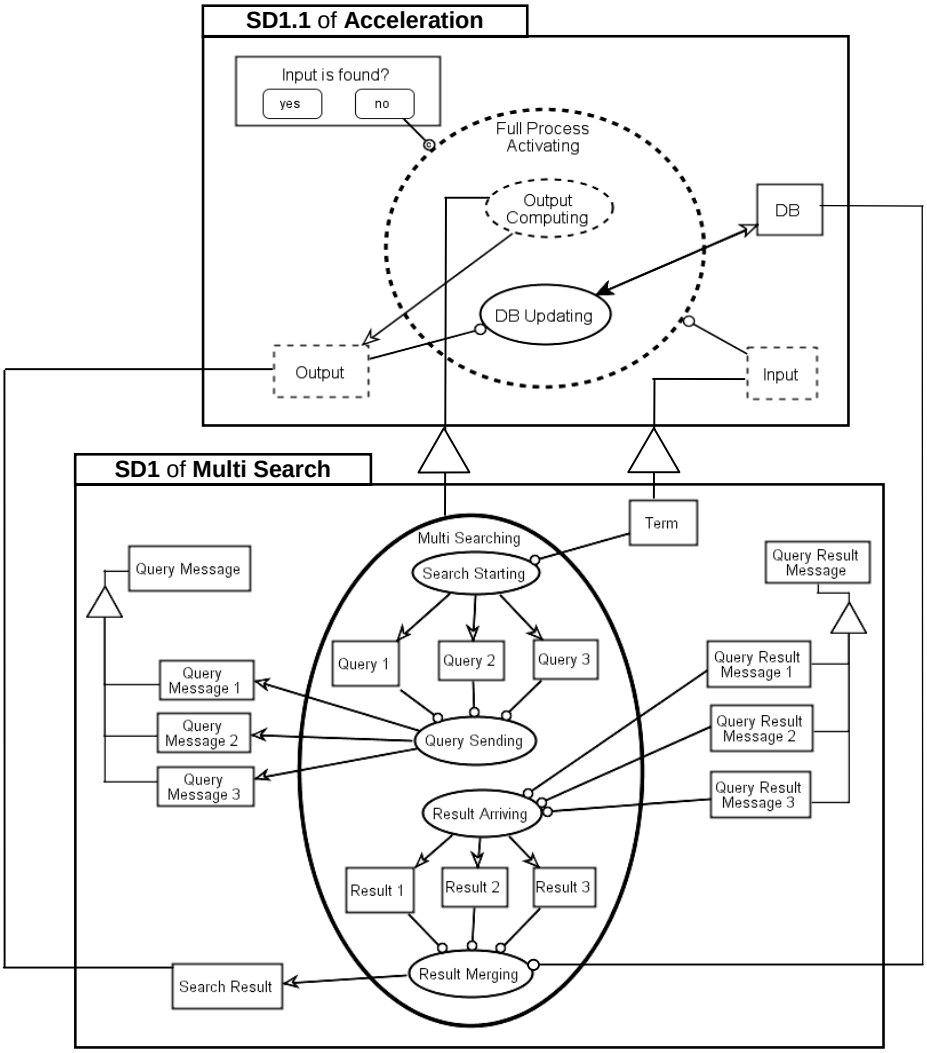

Figure 4. Improving the **Result Merging** algorithm of the **Accelerated Multi Search** module by linking it to **DB**.

Figure 5 shows how the **Accelerated Multi Search** module is enhanced with the ability to maintain a log file. The generic **Log Recording** module includes a systemic **Log File** along with its **Log Records** and a **Recording** operation. The only environmental thing in this module is the object **Input**. When weaving the **Log Recording** module into the **Accelerated Multi Search** module, **Term** is bound to **Input**. The two modules are also connected with an event

link and an invocation link, denoting the two possible triggers of **Recording**: a **DB** change event triggers **Recording** via the event link, while a **Multi Searching** process termination event triggers **Recording** via the invocation link. The double line arc between the two events represents a "logical-or" relation between them.

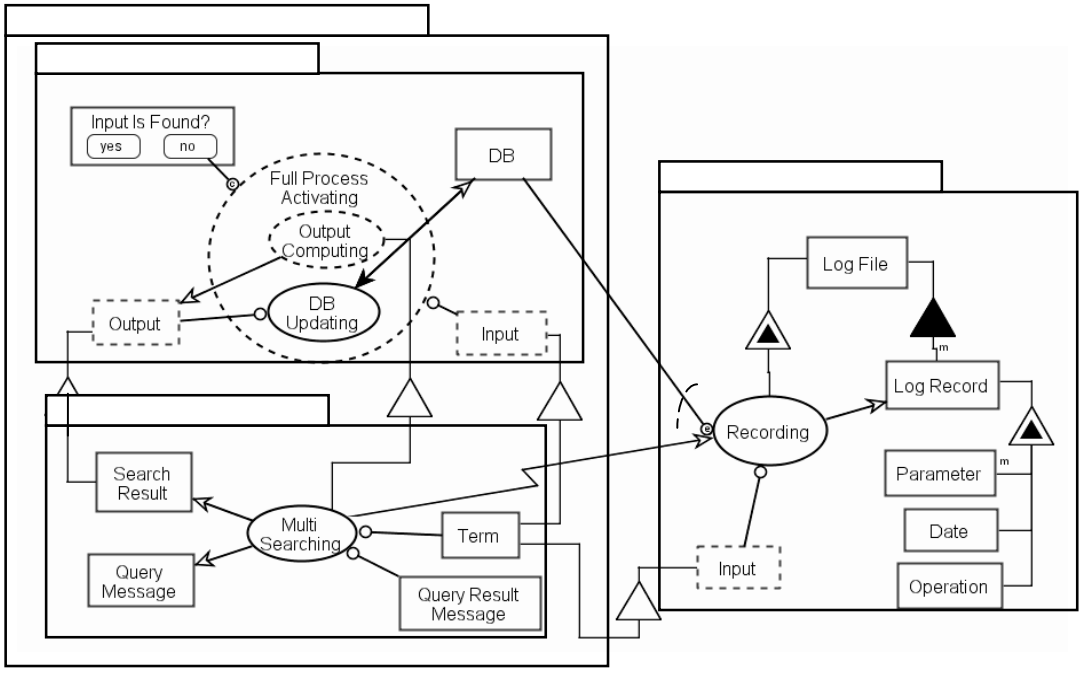

Figure 5. Reusing the **Log Recording** module in the **Accelerated Multi Search** module.

References

1 Firstenberg, Y., Katz, S. and Shmueli O. An Object-Oriented Program Accelerator Using Impersonation, Technion Computer Science Department Technical Report, CS-2002-06, 2002.