The article begins by acknowledging the transition from the older UML 1.4 specification to the new UML 2.0 draft that had been adopted by the OMG standards body at the time of writing. It outlines the key motivations behind UML 2.0 - enhancing UML's model-driven capabilities to better support Model Driven Architecture (MDA), improving scalability for modeling large applications, and enhancing the notation's readability, especially for tricky concepts like modeling logical flows which were challenging in UML 1.x.

The purpose and applications of sequence diagrams are explained next. As one of UML's essential behavioural diagrams, sequence diagrams are invaluable for sequentially documenting interactions between objects/roles across various domains - from high-level business process workflows to low-level technical software design. They can be used to document the current behavior of an existing system ("as-is") or specify the desired behavior and requirements for a new system implementation.

The article then introduces a new diagram notation in UML 2.0 - the frame element. This provides a consistent outer boundary and partitioned header/body structure for laying out diagrams. For specific diagram types involving interactions like sequence diagrams, the frame allows connecting incoming/outgoing messages directly to the frame edges.

The core sequence diagram notation elements are covered in detail:

Lifelines - These represent either specific object instances with underlined instance names (e.g. student: Student) or general roles/types without underlining. Conventions for naming lifelines representing anonymous instances are explained.

Messages - The solid arrowheads represent synchronous call operations, while stick arrowheads depict asynchronous signals between lifelines. Proper formatting and placement of method names, argument values, and return values along the message arrows are illustrated.

Self-calls - The ability to model scenarios where an object sends a message to itself, connecting the message arrow back to the same lifeline.

Guards - These conditional expressions placed above messages allow qualifying when a message should be sent based on factor values.

The limitations of the basic UML 1.x notation are then highlighted, especially the lack of advanced constructs to effectively model complex control flows like conditional branching or repetitive loops. This motivated new operators in UML 2.0 called "combined fragments" which provide a powerful way to visually model a variety of control flow scenarios:

Alternatives (alt) - For depicting mutually exclusive alternative flows akin to if-then-else logic. The article shows the framed notation partitioning each alternative flow and rules around guards.

Options (opt) - For depicting conditional flows that may optionally execute based on a guard condition (if-then scenarios)

Loops - For depicting repetitive message sequences based on Boolean conditions or procedural iteration constraints like minimum/maximum loops.

The article uses step-by-step explanations along with helpful visual examples to illustrate reading and interpreting each of these control flow notations.

Next, it covers some more advanced UML 2.0 features:

Interaction Occurrences (ref) - This powerful construct allows embedding references to other sequence diagrams, conceptually treating them as subprograms. The article shows the framed ref notation, conventions for specifying parameters/arguments, and rules around lifeline continuation across the referenced diagram.

Gates - An alternative mechanism to interaction occurrences for depicting data passing between sequence diagrams using message arrows connected to the frame edges.

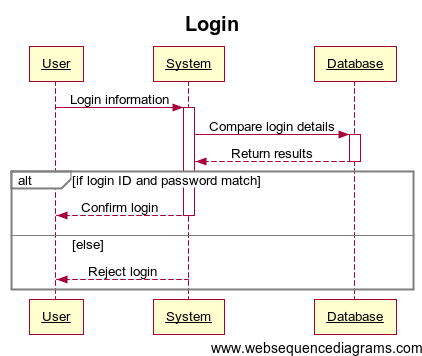
In conclusion, through its exhaustive coverage, copious examples, comparative analysis with UML 1.x, and expert viewpoints, this article serves as an authoritative and comprehensive tutorial on leveraging the significantly enhanced sequence diagram notation in UML 2.0 to effectively model and communicate interactions, control flows, and behavioural scenarios ranging from high-level requirements to low-level design in modern software/systems.

FIG 1 : USER LOGIN SEQUENCE