Seeing the software world from a dependency perspective

EurekAlert!

Software development is a complex and difficult task. Software developers and researchers try to deal with software development in a simple way from multiple perspectives. This leads to the use of various kinds of models, including informal, semi-formal, and formal models, and all kinds of development methods, including informal and formal methods. In fact, every software development method contains multiple models from different perspectives. In contrast to an informal method, a formal method is considered to be a set of tools and notations (with formal semantics) used to specify unambiguously the requirements of a computer system, and which supports the proof of properties of the specification and proofs of correctness of an eventual implementation with respect to the specification.

However, critics believe that a formal description has to take a very specific form depending on the formal method used. Moreover, existing formal methods adopt formal models that have well-known limits, e.g., pure mathematics and its complex notations and rules. Thus, to deal with these problems, Dr. Jianmin Jiang and his colleagues proposed a novel model, referred to as the protocol structure (also called a dependency structure) from a dependency perspective. They are of the opinion that all kinds of dependency relationships exist between the same or different things and that if there are no dependency relationships, the things are independent. These dependencies are divided into the following four classes: transformation, synchronism, choice, and priority. The classification of dependencies aims to distinguish causality, conflict, parallel, and priority dependencies. The idea was motivated by the definition of composition operations in process algebra and the event dependency relationships (causality and conflict) of event structures. Event nodes, i.e., events with data information, are regarded as basic elements. Therefore, the proposed model includes the following: (1) causality dependencies, which are called transformation dependencies, (2) choice dependencies corresponding to conflict dependencies, (3) synchronism dependencies, which are introduced to realize data merging and the joining of control flows, similar to token fusion in Petri nets, and (4) priority dependencies, which are used to enforce scheduling policies. If none of the above mentioned dependencies exists between event nodes, such event nodes are independent (parallel).

The novel model is represented as a tuple where M is a non-empty and finite set of event nodes, I is the set of initially available event node sets, T is the transformation relation, S is the synchronism relation, C is the choice relation, P is the priority relation, Out is the set of event node sets at output interfaces, In is the set of event node sets at input interfaces, and F is the set of finally available event node sets. This model combines several features of partial orders and Petri nets.

Specifically, in a distributed system, e.g., a service-oriented system, the interaction of components is realized through the exchange of messages. Then, the dependency relationships of messages can be used to model components and their interactions. Service-Oriented Computing is a new computing paradigm that utilizes services as fundamental elements for developing distributed and service-oriented software systems. Services, which can be logically or geographically distributed, are independent components. This distributed nature introduces new problems, because independent services need to collaborate to achieve a common goal. One of the new problems is communication. To formally analyze and verify service-oriented systems, most existing formal frameworks rely on synchronous communication. In fact, asynchronous communication is as realistic as synchronous communication, but results are more complicated to obtain and are sometimes undecidable. These existing attempts usually adopt mainstream formal models, such as process algebras, Petri nets, and automata-based models, for specifying services and communication between services. These mainstream formalizations also rely on the assumption of a communication model.

Dr. Jianmin Jiang and his colleagues specify synchronous communication, asynchronous communication, and broadcast communication in a uniform way using dependencies. Under synchronous communication, a synchronous message is divided into a sent message and a received message. Accordingly, a received message depends on its corresponding sent message. Under asynchronous communication, an asynchronous message is divided into three messages: a sent message, a received message, and a stored message. The stored message buffers its corresponding sent message. Accordingly, a received message depends on its corresponding stored message and the stored message depends on its corresponding sent message. Under broadcast communication, multiple receivers may also directly or indirectly depend on their corresponding sender through messages. Dr. Jianmin Jiang and his colleagues' work, entitled "Modeling and analyzing mixed communications in service-oriented trustworthy software", was published in SCIENCE CHINA Information Sciences.2012, Vol 55(12). Note that a message is an event with data information; that is, the occurrence of an action that a system may perform as described in the paper above.

A protocol (dependency) structure represents systems (including both software and hardware systems) in the following way. An event with data information is called an event node. The data information is not a concrete value but an abstract symbol corresponding to all possible values. The set of event nodes contains all the events involved in a system. The initially available event node set denotes the event node set that is available before the system starts executing. Transformation dependencies represent the causality relations of event nodes. Synchronism dependencies represent the joining or merging of multiple event nodes. Choice dependencies model the mutually exclusive relationship of event nodes, and priority dependencies control the order of occurrences of events in the system. The event node set at an output interface comprises the event nodes sent by the system to the environment, while the event node set at an input interface comprises the event nodes sent by the environment to the system. The finally available event node set denotes the event set available when the system, or its sub-processes, terminates.

A protocol (dependency) structure is a promising formal framework for describing and studying software systems, characterized as being concurrent, asynchronous, distributed, and parallel.

Source: http://www.eurekalert.org/pub_releases/2013-02/scp-sts020513.php [1]

Links:
[1] http://www.eurekalert.org/pub_releases/2013-02/scp-sts020513.php