01 Apr 2008

An Executable System Architecture Approach to Discrete Events System Modeling Using SysML in Conjunction with Colored Petri Net

Renzhong Wang

Cihan H. Dagli
Missouri University of Science and Technology, dagli@mst.edu

Recommended Citation

R. Wang and C. H. Dagli, "An Executable System Architecture Approach to Discrete Events System Modeling Using SysML in Conjunction with Colored Petri Net," *Proceedings of the 2nd Annual IEEE Systems Conference, 2008*, Institute of Electrical and Electronics Engineers (IEEE), Apr 2008.
The definitive version is available at https://doi.org/10.1109/SYSTEMS.2008.4518997

SysCon 2008 – IEEE International Systems Conference
Montreal, Canada, April 7–10, 2008

An Executable System Architecture Approach to Discrete Events System Modeling Using SysML in Conjunction with Colored Petri Net

Renzhong Wang, Cihan H Dagli
System Engineering Graduate Program, Missouri University of Science and Technology
600 W 14th Street, Rolla, MO, USA, 65409-0370, Email: rwkb4@mst.edu, dagli@mst.edu

Abstract *– This paper proposes an executable system architecting paradigm for discrete event system modeling and analysis through integration of a set of architecting tools, executable modeling tools, analytical tools, and visualization tools. The essential step is translating SysML-based specifications into Colored Petri Nets (CPNs) which enables rigorous static and dynamic system analysis as well as formal verification of the behavior and functionality of the SysML-based design. A set of tools have been studied and integrated that enable a structured architecture design process. Some basic principles of executable system architecture for discrete event system modeling that guide the process of executable architecture specification and analysis are discussed. This paradigm is aimed at general system design. Its feasibility was demonstrated with a C4-type network centric system as an example. The simulation results was used to check the overall integrity and internal consistency of the architecture models, refine the architecture design, and, finally, verify the behavior and functionality of the system being modeled.*

Keywords *– Discrete-event system, SysML, CPN, Modeling, Executable Architecture*

I. INTRODUCTION

Architecture modeling furnishes abstractions for use in managing complexities, allowing engineers to visualize the proposed system and to analyze the problem domain and describe and specify the architecture for the solution domain. However, most architecture packages still only produce static products. Static models are hard to verify and validate because in such models the collaborations between various components defined in the architecture and the information flows among them are specified in a static way. Consequently, they fail to depict the temporal relationships of those components as well as resource utilization over time and thus provide little information about how the system behaves in operational environments. For example, it is very hard, if not impossible, to explore causally chained events and possible system states given a trigger. Rigorous verification and validation of system specifications requires executable models. Simulation capability is typically integrated with executable architectures to further support dynamic analysis of system behavior, performance, and effectiveness.

The significance of executable modeling increase as systems become more complex. Many studies have been undertaken in this field, especially in the software industry. Among them, several schemes have been developed to make Unified Modeling Language (UML) executable, such as Executable UML (xUML), Executable and Translatable UML ($^{X}_{T}$UML), and Virtual Machines (VM). However, because their goal is automatic code generation, these approaches are all based on UML StateChart Variants, which means they take an asynchronous view of the system and focus on the reactive behavior of the individual object. For the purpose of general system modeling, the UML state machines lack well-defined executable semantics, do not support modeling of multiple instances of classes, and do not scale well to large systems.

An alternative approach to the executable architecture specification is to incorporate Colored Petri Nets (CPNs) as a supplement to UML diagrams. Currently, much of the work in this field is concerned with the transformation process [1, 2, 3]. Petri Nets have also been used to ascribe formal execution semantics to UML notations via a rule-based approach [4]. Some research even proposes a CPN profile for UML [5]. However, much of the work is still based on the transformation of UML state machines [1]. Only a few studies that emphasize the interactive behavior between systems components can be found in literature [6, 7]. Using only CPN to specify and simulate a system is also possible [8]. However, this method is not very common because CPN is not good at giving purely static descriptions of system architecture.

The MITRE Corporation developed an Executable Architecture Methodology for Analysis (EAMA) that translates DoDAF architecture into an executable form using a federation of business process models, communications network models, and combat simulations. Its primary application is in enterprise architecture. Still, relatively few studies [9] can be found that derives executable models for general system from System Modeling Language (SysML) specifications. The research described in this paper will contribute to this field of study.

The paper is organizes as follows. Section II discusses the methodologies that supports executable architecting paradigm. Section III presents their application to the modeling and analysis of the Global Earth Observation System of Systems (GEOSS). Finally Section IV sums up the conclusions and discusses directions for further research. The reader is assumed to be familiar with the basic ideas of SysML and CPN.

II. PROPOSED APPROACHES

A. Executable System Architecting Paradigm

The executable architecting is not yet a mature field. No single modeling tool currently available comes close to

978-1-4244-2150-3/08/$25.00 ©2008 IEEE

supporting the full range of capabilities needed for executable architecting (e.g. specification, presentation, simulation, and analysis of both the static structure and dynamic behavior of a system). Therefore, this paper proposes a combined use of several related tools in an effort to take the immediate advantage of the best features of each tool. The interoperability of these tools is, therefore, required and studied.

For the specification of formal models, the Systems Modeling Language (SysML) is preferred because it supports the development of a broad range of systems thanks to its rich set of diagrams, rigorous syntax and semantics, and easiness to interpret. Extended from UML, SysML is an object-oriented modeling language so it shares the same primitives and basic concepts with many other object-oriented modeling languages, which provides a basis for model interoperability. However, SysML is weak in executable semantics, which limits its capabilities to analyze and verify defined specifications.

Formal specification of the executable model requires well defined executable semantics. The chosen of the modeling language depends on the system to be modeled, the abstraction level to work on, and the system behavior of interest. In many modern engineering systems such as communication networks, flexible manufacturing systems, control systems, transportation systems, and C4 systems, the behavior of interest is driven only by events that occur at discrete time points. Such systems can be best specified by discrete-event models. As defined in [10], discrete-event models represent the operation of a system as a chronological discrete sequence of events. Each event occurs at an instant in time and marks a change of state in the system. Therefore, an executable architecture specified in this way is a dynamic model that defines the precise event sequences, the conditions under which event is triggered and information is produced or consumed, and the proprieties of producers, consumers and other resources associated with the operation of the system. Usually, the complexity of such systems stems from the fact that the overall system behavior is not only determined by the components individually but also from their interactions. Therefore, the target system should be modeled as a collection of objects and their interconnections, information to be processed and exchanged, the order of events, and other properties.

A variety of executable formalisms have been developed that support the development of discrete-event systems and offer the capabilities for dynamic behavior analysis, for example, Finite-automata, StateCharts, DEVS (Discrete Event System Specification), Petri nets, and GSMP (Generalized Semi-Markov Processes). The approach proposed in this paper intends to accommodate as broad a range of systems as possible. Hence, we are interested in a modeling formalism that is sufficiently general, i.e. independent of domain and technological substance of a specific system, and easy to map to selected formal model specifications, i.e. SysML. More specifically, we want an executable modeling formalism that is based on generic dynamic systems concepts, i.e. states and transitions, supports concurrency, synchronization and resource sharing, offers hierarchical description and modularity, and facilitates analysis and verification. Based on these criteria, the Colored Petri Net (CPN) emerges as the best choice among those formalisms investigated. The basic notation for Petri Nets is a bipartite graph consisting of places and transitions that alternate on a path and are connected by directional arcs [11]. Tokens are used to mark places, which represent the state of a system. When certain conditions hold, transitions will be fired, causing a change in the placement of tokens and thus the change of system states. CPNs extend the vocabulary of basic Petri Nets allowing tokens to have an associated attribute and add new features that enable Petri Nets to scale to large system modeling. CPNs combine the strength of ordinary Petri Nets with the strength of a high-level programming language, which provides the primitives for definition of data types and manipulation of their data values [12]. Reference [12] provides an in-depth discussion of the advantages of using CPN. What also needs to be mentioned is that three characteristics distinguish CPNs from other executable formalisms. First, CPNs offer an advantage of combining a well-defined mathematical foundation, an interactive graphical representation and simulation, and the capabilities to carry out simulations and formal verifications. Secondly, it is possible to use the same (or at least very similar) models to check both the logical or functional correctness of a system and for performance analysis [15]. Finally, CPNs are very flexible in token definition and manipulation. Various architectural elements, e.g. components, tasks, messages, events, and even use cases can all be described by different types of tokens. This feature makes CPN modeling even more flexible and capable of modeling a large variety of systems.

Formal models specified by SysML can be transformed to executable models represented by CPNs by following well defined procedures and mappings between these two notations. This transformation supplements SysML modeling with formal dynamic semantic plus the behavioral modeling and analysis strength. CPNs have a formal, mathematical representation, which not only can unambiguously define the behavioral properties but also forms the foundation for formal analysis methods. Information about the structure and simulation of a CPN can easily be extracted and communicated with external applications and processes, which provides a means to enhance simulations and further extend CPN's capabilities in model analysis with the aids of other analysis tools.

In summary, by integrating the above mentioned tools, we can create an executable architecture paradigm that offers a structured design process as shown in Fig. 1. This is an iterative process starting with requirements analysis and specification, through which the desired behavior of the system is captured. The executable model (represented by CPN) developed from the static model (a set of SysML diagrams) is capable of generating dynamic behavior (the behavior as modeled). Key information can be extracted from the simulation to support architecture evaluation and analysis. Based on the results, the system can be modified and another design cycle can begin. Finally, by comparing the desired behavior and the behavior as modeled, we are able to verify the system architecture being designed.

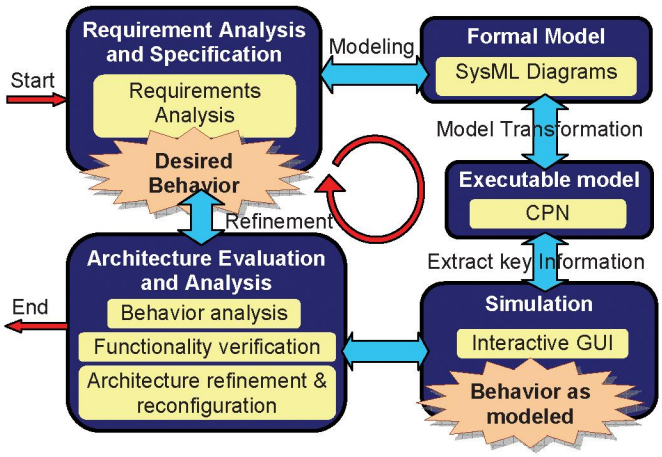

Fig. 1. Executable System Architecting Paradigm

B. Transformation from SysML to CPN

Pre-conditions. The architecture specification should be formal enough to accommodate executable semantics. That is, it must capture sufficient representations of architectures and be unambiguous and consistent. An architecture will not be fully operational until all nodes and activities are properly configured and connected and are consistent in terminology, definition, and data exchange syntax.

Transformation schemes Based on Static View versus Dynamic View. The goal of developing an executable architecture model in this paper is to facilitate the investigation of system wide properties. Because the interactive behavior of system components is of greater interest than the reactive behavior of individual components, it is better to define the executable model that relies on synchronous operation calls between operational nodes. For this reason, the SysML-to-CPN transformation discussed in this paper is primarily based on SysML sequence diagrams.

Basic mapping from SysML models to CPNs.

Places/Transitions. There exist two basic alternatives to map from the SysML specifications to the CPNs resulting in two different interpretations:

1. Identify actions with places in the CPN. In this case, the state of a system can be interpreted as what the system is doing. This is the way that UML/SysML state machine usually adopted, which is good for depicting the reactive behavior of a system.
2. Identify actions with transitions. In this case, the state of the system can be interpreted as a set of conditions that a system is holding and a set of effects after the system did something.

The second alternative is chosen in this paper for the following two reasons. First, the hierarchical (or modularity) of CPN is achieved by means of substitution transitions. By identifying actions with transitions, we can decompose the action using substitution transition to further elaborate on that behavior or decompose the entity that generates this action. Secondly, modeling actions by transitions allows us to model data flow and/or control flow more clearly. This is desirable since the essence of many discrete systems is information processing.

Actors: Since, for every action, there is an actor that is responsible for that action, the active objects (objects having an action) in SysML can be bound to transitions.

Tokens. When places are used to model conditions and effects, tokens can be used to model resources, control signals, and input/output message or other entities to be processed or exchanged during a transition (action).

Transformation procedures. The transformation from SysML specifications to CPNs must be faithful for the simulation of the executable model to be used to verify and validate the SysML model. Accordingly, an unambiguous mapping between the elements of various SysML diagrams and CPNs must be established. Fig. 2 outlines the procedure used in this paper for synthesizing a CPN model from a SysML model. Note that, in order to facilitate simulation and performance analysis, some extra CPN constructs such as simulation monitors, which are not converted directly from the SysML model, are allowed to be added to the original CPN model, provided that the logic of the system is not impacted.

Step 0: Augment the sequence diagram(s). For each object in the sequence diagram(s), add the operation description to the appropriate position on the lifeline in between the input and output message/event. The operations should have been defined in block definition diagram(s)

Step 1: Create a transition for each operation in the sequence diagram(s) (preferably also list the object description next to the operation description).

Step 2: Create a substitution transition for each nested sequence diagram.

Step 3: Create a place for each message/event between lifelines. Assign the appropriate color set and create the corresponding declaration in the index.

Step 4: Create arcs between transitions and the places according to the sequence diagrams. There should be a one-to-one matching between the numbers of message/event in the sequence diagrams and the number of places between transitions in the CPN model.

Step 5: Add Arc inscriptions, guard functions, or code segments derived from the rules associated with each operation.

Step 6: Create a sub-page for each substitution transition.

6.1. Follows step 0 to 5 to create all the related transitions, places and arcs.

6.2. Assign the Input, Output, and I/O ports places.

Step 7: Assign socket places and connect all substitution transitions and their sub-pages.

Step 8: Specify initial markings for each related places.

Fig. 2. Translation Schemes from SysML to CPN

Based on the procedure outlined in Fig. 2, the basic mappings between elements in SysML diagrams and elements in a CPN model are generated and presented in Table 1, which

also establishes a concordance between various entities within a set of SysML diagrams.

Table 1. Mapping between Elements in a SysML Model and a CPN Model

System Entities	Elements in SysML Diagrams			Elements in CPN Model
	Sequence Diagram	Activity Diagram	Block Diagram	
Active object	Interacting Object	N/A	Part	(Substitution) transition
Passive / connector object	N/A	N/A	Interface	Place
Transient information /event	Information on the message line between lifelines	Object flow	Item flow	Token
Message Type	N/A	N/A	Flow specification	Place and its color set declaration
Operation Call	Information on the message line and/or description line	Object flow	Interface specification	Token
Operations	N/A	Action	Block definition	Transition
Flow	Message line between lifelines	Dashed line connecting Object flow and action	Port and Port connection	Arc
Module	Nested sequence diagram	Child activity diagram	N/A	Substitution transition

C. Integration of CPN with Supporting Tools

The CPN modeling language is supported by CPN Tools, which is a graphical software tool for creating, editing, simulating and analyzing CPN models. CPN Tools provides Comms/CPN, a CPN ML library, which allows CPN Tools to communicate based on TCP/IP with external application and processes, which provides a means of extending CPN's simulation capabilities, e.g. extraction of useful information, Graphic User Interface (GUI), instant feedback, and interactive control of the simulation process. For example, two GUIs are often used with a CPN, BRITNeY Suite [13] and Graphviz. The BRITNeY Suite is a java application that can run on top of CPNs. During the simulation, users can control the simulation execution only through this GUI. A variety of graphic outputs such as the Message Sequence Charts (MSCs) and the State Space Graphs can be generated after the simulation. They are important means for analyzing the behavior of the system being modeled. Graphviz is another option for generating various graphical outputs from CPN simulation, e.g. State Space Graphs. More software tools supporting the analysis of CPN can be found in [14].

D. Architecture Analysis and Evaluation

CPN models and simulations contain detailed quantitative information about the performance of a system, such as throughput, processing time, queue lengths, resource utilization, etc., which can be extracted to support the investigation and discovery of structural and dynamic system properties, which reflect correctly the behavior of the system in reality.

Three forms of architecture evaluation, logical, behavioral, and performance, are described in [7]. The logic is examined by testing each step of the execution to ensure that the model follows the desired logic. The behavior of the system can be observed directly from the simulation. However, it is often beyond the capability of human beings to observe the details of a simulation by watching the CPN and its markings. A numbers of alternative behavioral analysis methods are provided in [14] such as simulation report, report places, business charts, Message Sequence Charts (MSCs), state space reports (dead transitions, liveness, home properties, deadlock, conservation properties, etc.), and state space graphs. Reference [15] provides an overview of some new performance analysis facilities supported by the latest version of CPN Tools.

Behavior and Functionality Verification. When the whole set of conditions and events of a system are specified correctly in a CPN model, the model should be able to undergo appropriate sequences of state transitions. Therefore, we can verify the system design by comparing the behavior as modeled and the desired behavior. The former can be obtained from the Message Sequence Charts (MSCs) while the latter are captured by the sequence diagrams. If the comparison shows a match, the model can be verified and validated. If the match is insufficient, then either the architecture model needs to be modified in order to better represent the system architecture or the system architecture needs to be reconfigured in order to better satisfy the requirements.

Identification of Missing Specifications and Missing Requirements. Missing specifications can be identified in the process of both executable model synthesis and simulation because an incomplete model is not executable. Simulation runs can also reveal missing requirements, which, in this context, are functions or capabilities that the system must support in order to generate the required behavior or performance but have not been specified yet.

III. APPLYING EXECUTABLE ARCHITECTURE PARADIGM TO GEOSS

The Global Earth Observation System of Systems (GEOSS) is a system of networked sensors, communication devices, storage devices, computers, and other resources used in concert to observe the Earth. In this paper, GEOSS was modeled as a distributed multi-task concurrent information processing system with high interoperability, maintainability, and

expandability. The challenge is to model the management, retrieval, and processing of the observation datasets and information products in a distributed and heterogeneous computational environment that links distributed centers, users, data, applications, computer networks, and storage resources.

A system design that is resilient to change is highly desirable. Hence, the Model Driven Architecture (MDA) approach [16] was employed to guide the architecture development process. The MDA approach enables the same model specifying business processes or application functionality to be realized on multiple platforms. The benefit is great improvement in portability, interoperability, reusability, and maintainability. In general system design, MDA can be achieved through an iterative refinement process, which is driven by introducing domain information such as structural, behavioral interoperability, and interfacing requirements of system components. The resulting system architecture was a layered architecture rather than the typical federated one. This style of organizing the components standardizes the architecture while greatly leveraging flexibility. Fig. 3 is a SysML block definition diagram showing the relationships of various components within GEOSS. The system activities are realized as five layers and a cross-cutting section based on their roles in data and information processing. Lower layers provide service to upper layers and upper layers are logically closer to end users.

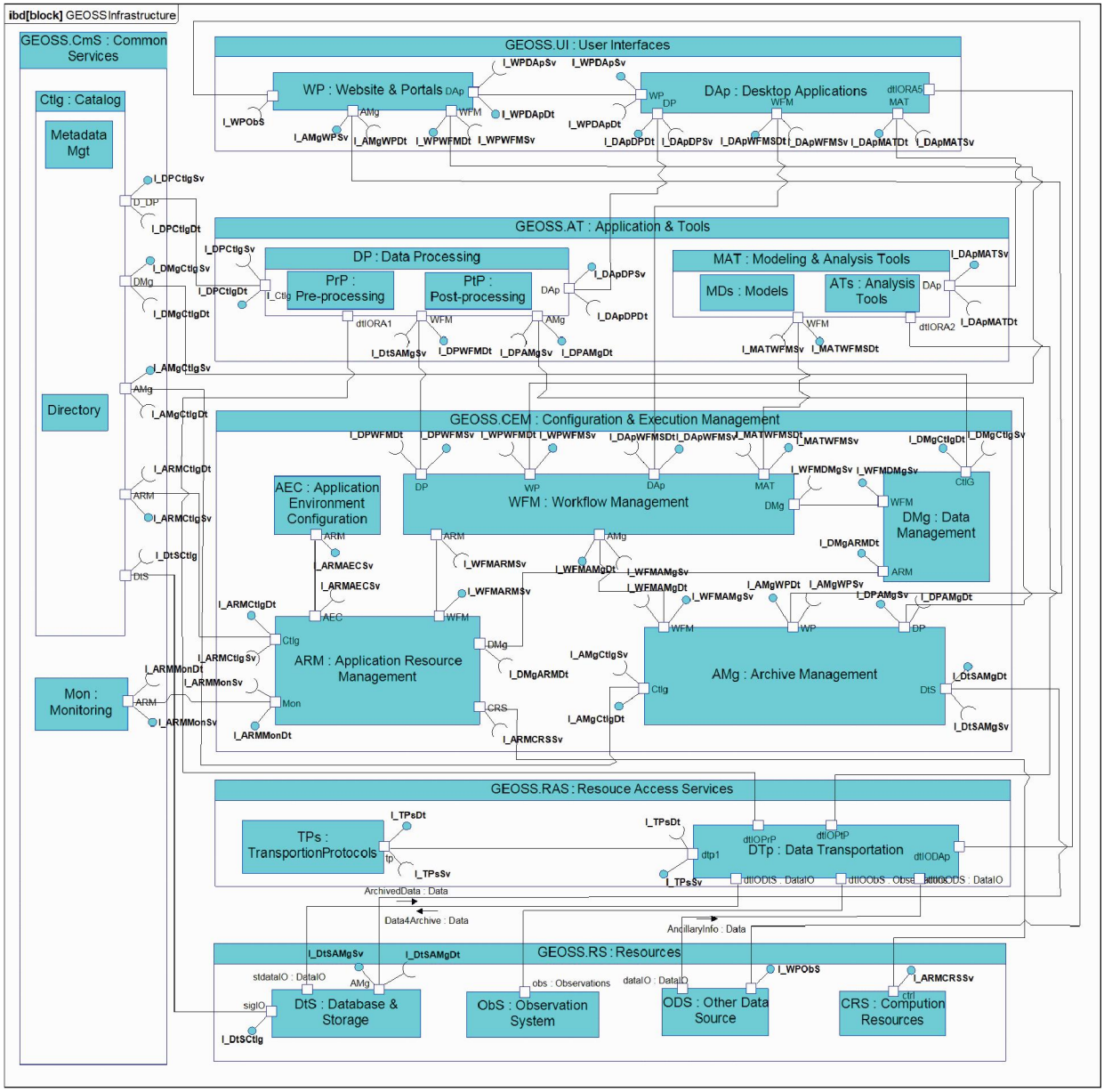

Fig. 3. Internal Block Diagram - GEOSS Internal Connections

Layer 1, the User Interface, comprises components that interact directly with end users and end-user tools. Layer 2, Applications and Tools, comprises common applications and tools that provide services to user applications. Layer 3, Configuration and Execution Management, comprises "service modules" that mange distributed resources such as application environment configuration, distributed computational resources coordination, and application input and output, archives and workflow management. Layer 4, Resource Access, provides the data transmission service and the standard protocols for accessing raw services. Layer 5, Resources, represents all the physical raw resources such as distributed database and storage, computational hardware and software, sensors, and data collection centers. Some cross-cutting components providing functionality that spans multiple layers are identified and grouped into a package called Common Services.

The behavior of the system is specified using SysML activity diagrams and sequence diagrams. Since the SysML-to-CPN transformation methods developed in this paper is primarily based on the latter, the example shown here only includes sequence diagrams. Fig. 4 depicts the sub-activity of collecting observation data.

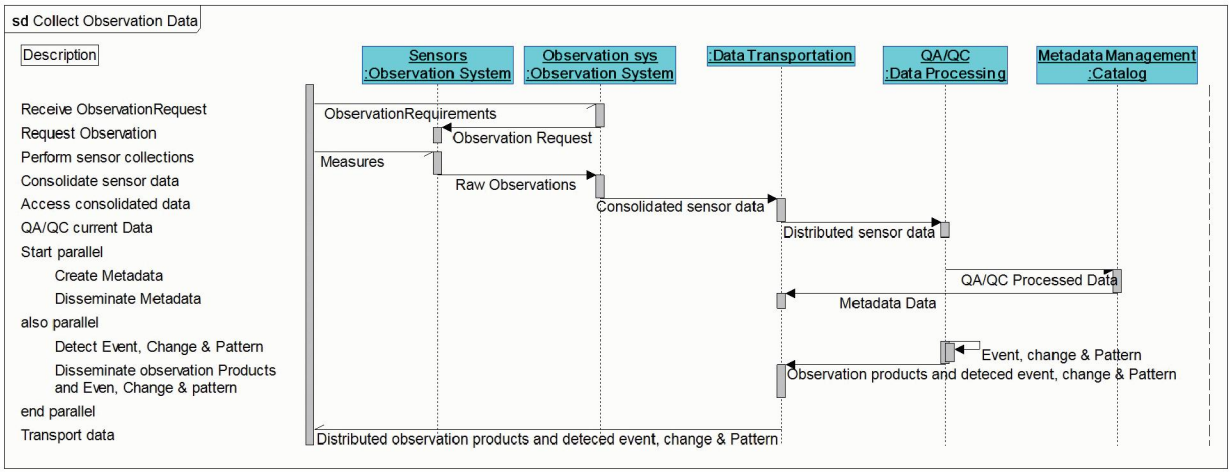

Fig. 4. Sequence Diagram – Collect Observation Data

The MDA principle fosters modularity which can be reflected through nested sequence diagrams. These modules, achieved through CPN substitution transitions, can be developed and tested in isolation. Fig. 5 depicts the CPN module derived from the sequence diagram presented in Fig.4 using the transformation rules defined in Section II.

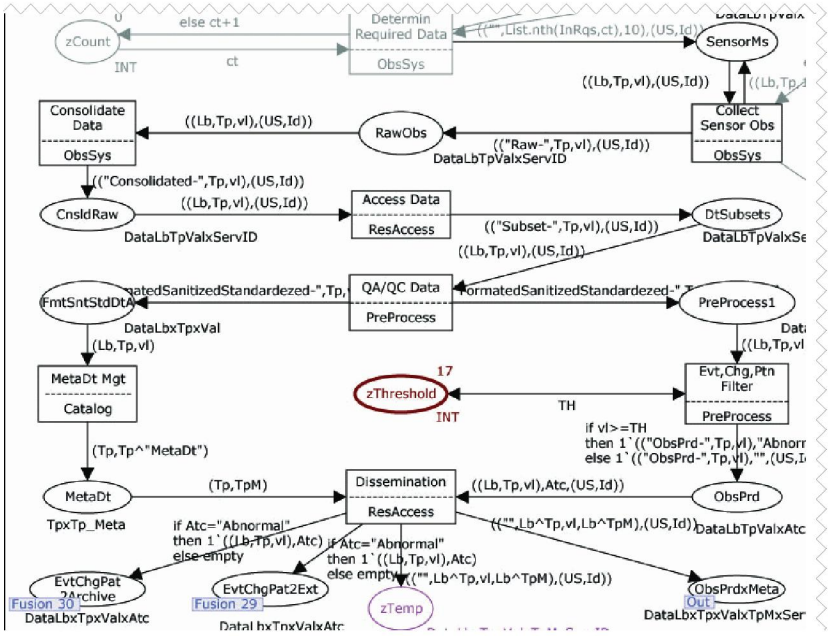

Fig. 5. CPN page – Collect Observation Data

Three animation tools supported by BRITNeY have been used in this paper:

1. Interactive Control, which includes accepting inputs from outside users and providing graphical feedback,
2. Message Sequence Charts (MSCs), and
3. State Space Graphs.

The MSC generated by the BRITNeY Suite from one simulation run of the above CPN model is shown in Fig. 6.

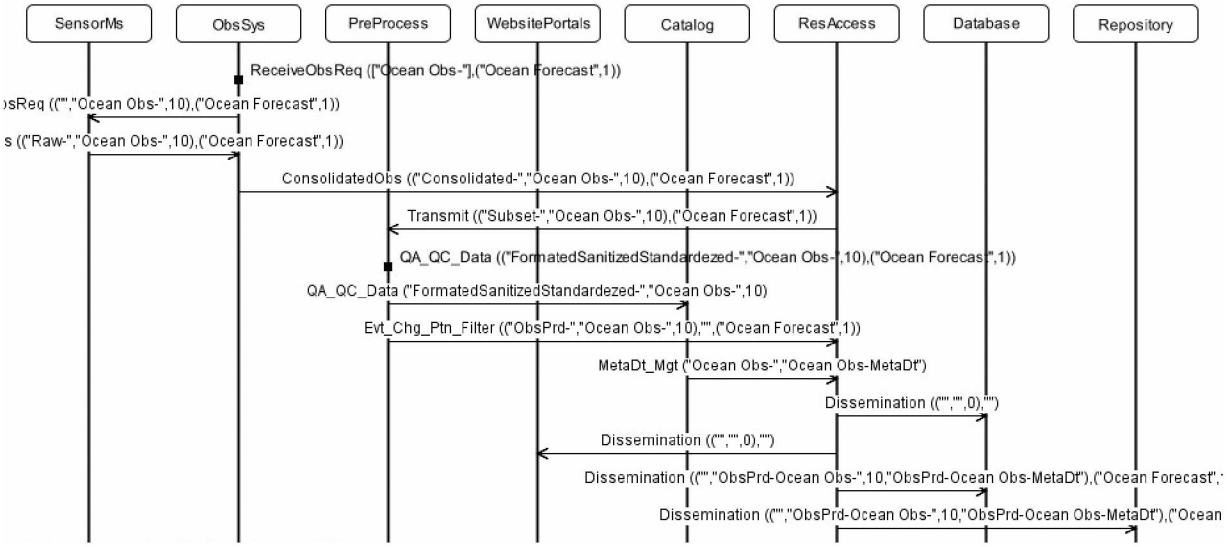

Fig. 6. MSC – Collect Observation Data

By comparing the above SysML sequence diagram and the corresponding MSC, we can conclude that the behavior as molded (reflected by MSC) conforms to the desired behavior (captured in the sequence diagrams). Thus, the system architecture can be verified.

IV. CONCLUSION AND FUTURE WORK

This paper introduced an executable system architecting solution based on SysML-CPN transformation. The approach proposed here models interactive behavior between various system components using states and transitions, as well as conditions and events, as the core semantics. To achieve this framework, a set of methodologies including a formal transformation procedure, a well defined mapping between these formalisms, and some architecture analysis technologies were developed. This paradigm facilitates the investigation of system design before the implementations starts. The benefit is an improved understanding of key design issuers, fewer design errors, and faster system development. The feasibility of this paradigm has been demonstrated using an information system as an example. This methodology should be able to generalize and be applied to a broad range of discrete-event driven concurrent system designs.

In this paper, the modeling activities emphasize the functional aspects of the system. Nonfunctional performance, such as time related, resources related, optimization, scheduling, security, and reliability, may be more of our interest sometimes. Quite often, non-functional performance is emergent behavior and thus simulation plays an even more important role in performance forecasting, evaluation, and verification. Further study need to be carried out in this area. Non-functional concerns are often coupled. For example, resource constraints may impact processing time and cause task scheduling and prioritization problems. Non-functional requirements can also impose constraints on the functional behavior. For example, security requirements may require the system to provide registration, subscription, authorization, or authentication services while accessibility may require resource control and prioritization capabilities. In order to simulate and measure the non-functional performance, some mathematic methods, computation intelligence tools, and other external simulation environments may need to be integrated into the executable model. More analysis techniques should be studied and integrated.

REFERENCE

[1] Hu, Zhaoxia, and Sol M Shatz. "Mapping UML Diagrams to a Petri Net Notation for System Simulation." In *Proceedings of the International Conference on Software Engineering and Knowledge Engineering (SEKE), Banff, Canada, 2004.*

[2] Pettit, Robert. G. and Hassan Gomaa. "Validation of Dynamic Behavior in UML Using Colored Petri Nets." In *Proceedings of UML 2000 Behavioral Semantics Workshop, York, England. October 2000.*

[3] Faul, Ben M. "Verifiable Modeling Techniques Using a Colored Petri Net Graphical Language." *Technology Review Journal*, (Spring/Summer 2004). Northrop Grumman, 2004. http://www.ms.northropgrumman.com/PDFs/TRJ/ TRJ-2004/SS/04SS_Faul.pdf (accessed June 2007).

[4] Baresi1, Luciano, and Mauro Pezzè. "On Formalizing UML with High-Level Petri Nets." In *Concurrent object-oriented programming and petri nets: advances in petri nets*, 276-304, New York: Springer-Verlag, 2001.

[5] Hansen, Klaus Marius. *Towards a Coloured Petri Net Profile for the Unified Modeling: Issues, Definition, and Implementation.* Technical Report COT/2-52-V0.1 (DRAFT). Center for Object Technology, Aarhus, Denmark, 2001.

[6] Bienvenu, Michael, P., Shin, Insub, and Alexander H. Levis. "C4ISR Architectures: III. An Object-Oriented Approach for Architecture Design." *Systems Engineering*, Vol. 3, No. 4, 2000. John Wiley and Sons Inc..

[7] Wagenhals, Lee, W., Sajjad Haider, and Alexander H. Levis. "Synthesizing Executable Models of Object Oriented Architectures." *Systems Engineering*, Vol. 6, No. 4, 2003. Wiley Periodicals, Inc..

[8] Kristensen, Lars Michael, Jens Bæk Jørgensen, and Kurt Jensen. "Application of Coloured Petri Nets in system development." In *Lectures on Concurrency and Petri Nets: Advances in Petri Nets*, Ed. Jörg Desel, Wolfgang Reisig, Grzegorz Rozenberg, Vol. 3098 of *Lecture Notes in Computer Science.* 626-685. Springer-Verlag, June 2004.

[9] Rao, Madwaraj, Sreeram Ramakrishnan, and Cihan H. Dagli. "Modeling and Simulation of Net Centric System of Systems Using Systems Modeling Language and Colored Petrinets: A Demonstration Using The Global Earth Observation System of Systems." *Systems Engineering*, unpublished.

[10] Banks, J. *Discrete-event System Simulation.* Pearson Prentice Hall, Upper Saddle River, NJ. 2005.

[11] David, René, and Hassane Alla. "Petri Net for Modeling of Dynamic Systems: A survey." *Automatica*, vol.30, no. 2. 175-202,1994.

[12] Jensen, Kurt. "An Introduction to the practical use of Colored Petri Nets." In *Lectures on Petri Nets II: Applications*, Ed. Wolfgang Reisig and Grzegorz Rozenberg, Vol. 1492 of Lecture Notes in Computer Science. 237-292, Springer-Verlag 1998.

[13] Westergaard, Michael, "BRITNeY Suite Home Page." University of Aarhus, Aarhus N, Denmark. http://wiki.daimi.au.dk/britney/britney.wiki (accessed Jan 2008)

[14] Jensen, Kurt. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use Volume 1: Basic Concepts.* Springer-Verlag, 1992. *Volume 2: Analysis Methods.* Springer-Verlag, 1995. *Volume 3: Practical use.* Springer-Verlag, 1997.

[15] Wells, Lisa. "Performance Analysis using CPN Tools." In *Proceedings of the Tenth IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'02), 2002*

[16] Miller, Joaquin. MDA Guide, Version 1.0.1, Ed. Jishnu Mukerji. Object Management Group, omg/2003-06-01, 12 June 2003. http://www.omg.org/cgi-bin/doc?omg/03-06-01 (accessed Jan 2008).