System Verification of Autonomous Underwater Vehicles by Model Checking

L. Molnar and S. M. Veres
School of Engineering Sciences
University of Southampton
SO17 1BJ, United Kingdom

***Abstract*- The paper addresses formal systems verification of autonomous underwater vehicles (AUV). The verification process includes hybrid system modelling and formulation of the discrete representation using natural language programming (sEnglish) via compositional abstraction - employing data, perception and action abstraction methods; the translation of the discrete abstraction into interpreted script programming language (ISPL) of a mainstream multi-agent model checker (MCMAS) for reachability verification of undesirable states, and ultimately the bridging into the discrete event system representation in Stateflow formalism. Using this technique, modelling and model checking can include the complete physical system of the autonomous vehicle, its multi-agent software on board and also the human interface represented as an agent.**

I. INTRODUCTION

AUV mission aborts are very costly. If a vehicle performs seafloor survey at a depth of 3000m and aborts its mission, then at 2m/s speed at 30 degrees rise it takes an hour to surface and several hours if surfacing by positive buoyancy. Adding to this the hours spent on diagnosis and repair the mission delay can be half a day at a cost of about £1000 per hour [1]. Part of the reason for this is that an AUV is an engineering platform that contains many different subsystems that were not designed to work together. The system engineer needs to design an inherently reliable vehicle from individual parts that are reliable in themselves. A mission is deemed successful when the vehicle surfaces at the end of its mission without any irrecoverable faults during the mission and, even more importantly, all data are stored onboard. Very few publications are available on reliability of AUVs, the two major studies available are [2] and [3]. Also a report of the Autosub Loss Inquiry Board [4] supports these findings. Reference [2] summarizes 14 years of experience with Autosub I and II [5], during which 240 missions were completed. The authors in [3] classify the cause of failures as:

(A) Hardware component failures (electronics hardware, mechanical problems, pressure vessel leaks, bad GPS, acoustic telemetry, power supply problems);

(B) Hardware component failure in combination with unusual external interference (acoustic interference, bad GPS);

(C) Vehicle operational errors (collision with vessel, software errors, failure to dive);

(D) Human operational errors during preparation, launch and recovery;

(A)-(D) classes mean different dimensions of the system that must be controlled to avoid failures. One can also classify failures by saying: (a) which subsystem failed (power, communication, control electronics, mechanical system, payload); (b) which functionality has been affected (maneuverability, descend, ascend, ability to perform mission steps, ability to provide functional environment for payload, etc.); (c) how severely the failure is (no action needed, human action needed, not critical, moderately critical, critical, unrecoverable vehicle).

In [6] a software (RECOVERY) is reported that offers practical self-diagnostics for AUVs. Enhanced environmental awareness [7], [8] of autonomous robots with suitable agent based programming can enhance the level of intelligent problem solving behavior in face of problems. These approaches are only effective after one actually has a technically reliable vehicle already. The research reported here investigates a methodology for reliable one-off design that includes evaluation of traditional engineering fields in combination with reactive agents used on AUVs that execute a sequence of mission commands. Reference [9] presents a systematic method of verification for an AUV as a hybrid system. A bottom-up approach is developed in which the bottom level of the hybrid system hierarchy is verified first, and each higher-level is subsequently verified with the assumption that all lower levels are correct. At each step in the verification process, lower and higher levels than the one currently being verified are abstracted, thus reducing the complexity of verification. They algorithmically developed and integrated into the design a hierarchical mission-level controller for an autonomous underwater vehicle. Reference [10] is concerned with the development of an unmanned vehicle (UV) ontology and automated conversion of vehicle specific information into a data format constrained by this ontology. This is useful by enabling the development of planning and analysis tools for arbitrary vehicles and enabling facilitating interoperability between dissimilar vehicles. Implementation of this ontology is carried out using the Extensible Markup Language (XML).

Reference [11] describes the validation and verification of the remote agent for spacecraft autonomy.

There is no overall reliability technology available that would be able to cope with all aspects of AUV reliability:

(1) The RECOVERY [6] systems offers a partial solution by enhancing survivability by on board diagnosis. The problem

1-4244-2523-5/09/$20.00 ©2009 IEEE

however remains that some failures still may happen and even the cleverest artificial intelligence (AI) system will not be able to rescue the situation.

(2) No system offers a satisfactory solution for the failsafe design and operation of AUVs by humans when operation includes the work of technical personnel who can make mistakes.

(3) Although subsystem robustness tests are performed there is no proper modelling and assessment of an assembled AUV with regards to the actual risks where operations of subsystems are interdependent.

(4) There is no systematic modelling approach available to understand the environment in which the AUV will operate.

This paper reports research to enhance the reliability of autonomous underwater vehicles, with a particular focus on the Autosub AUV (see Fig. 1). The paper is organized as follows. Section II presents the outline structure of the integrity and fault assessment system (IFAS) as a hybrid systems modelling problem. Section III describes the process of creating a discrete event model via discrete abstractions obtained by formulation with natural language sentences. Section IV and V shows how the discrete event system can be translated in a Stateflow chart for the purpose of simulation, respectively for verification in a multi-agent model checker system. Section VI illustrates some of the modelling and code involved before conclusions are drawn.

II. Integrity and Fault Assessment System

A waterfall model of the IFAS development process (see Fig. 2) is enumerated in the following:

1) A formal model is defined for the examined autonomous system.

2) The model is refined for adapting it to a mainstream model checker. The extensive refinement consists of a discrete abstraction of the hybrid system model and a conversion of the design into a formalism accepted by the model checking tool.

3) The requirements of the system are analyzed, hence asserting the properties that the design must satisfy. The specifications to be verified are formulated in the model checker's logical formalism.

4) The model checker is used to verify whether these requirements are satisfied or whether bugs exist. The consequences of partial or complete faults in physical components are also tested.

5) If the verification stage of the model checking produces counter examples (to demonstrate possible faults) consisting of an execution trace [12], then the design process is resumed to extend and/or to refine the modelled system.

Research efforts in the area of autonomous systems [13],[14] have shown that most successful approaches need to abstract out parts of the system. Avoiding the use of hybrid automatons in the modelling process (as a basis of modelling the overall system) is crucial as that helps to avoid the "curse of dimensionality" and therefore poor overall effectiveness of verification. We carry out modularization, where systems are composed of subsystems connected by finite state inputs and outputs, can be assessed for reliability on the basis of the reliability of their component subsystems. This way the total reliability of a complex system can be evolved from bottom up.

Various definitions of hybrid systems have been published in [15-19]. In the following a generic denotation is used.

A hybrid system or automaton is a tuple

$$H = (V, X, f, Init, Inv, E, Grd, R) \qquad (1)$$

where,

V represents a finite set of locations or discrete states;
$X \subseteq \mathbb{R}^n$ is a set of continuous states;
$f: V \times X \to 2^{\mathbb{R}^n}$ is a vector field of flow determining the continuous dynamics;
$Init \subseteq V \times X$ is the set of initial states;
$Inv: V \to 2^{\mathbb{R}^n}$ assigns to each discrete state an invariant set;
$E \subseteq V \times V$ is a set of edges;
$Grd: E \to 2^{\mathbb{R}^n}$ is a guard condition;
$R: E \times X \to 2^{\mathbb{R}^n}$ is a reset map.

Each discrete state has a set of initial states $\{x \in X | (q, x) \in Init\}$, a vector field $f(q, x) \subseteq \mathbb{R}^n$ and an invariant set $Inv(q) \subseteq \mathbb{R}^n$ associated with it. Starting from the initial value corresponding to the discrete states, the continuous state evolves in time satisfying the condition $\dot{x} \in f(q, x)$. The continuous state can evolve as long as it remains in the invariant set, i.e. in the domain of permitted evolution within a discrete state. When the continuous state x reaches the guard conditions $Grd_{q,q'} \subseteq \mathbb{R}^n$ of the edge $(q, q') \in E$, the switching of the discrete states occurs from q to q'; at the same time the continuous state is reset to a value in $R_{q,q'}(x) \subseteq \mathbb{R}^n$.

Fig. 1. Autosub 6000 of the National Oceanography Centre at the University of Southampton that can reach an ocean depth of 6km.

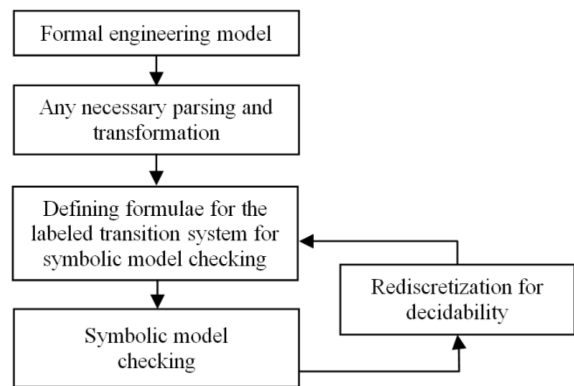

Fig. 2. IFAS development stages.

Finite discrete systems can be obtained from hybrid systems by abstraction. The abstraction process can help to produce a less complex but less detailed model of the hybrid system [20], while maintaining the properties of interest. If the properties of interest are preserved by bisimulation in the obtained finite discrete system, then the verification of this abstracted system should be equivalent to checking the same property on the original system [16].

A transformation or parsing process applied to an abstracted system may be necessary so that the system complies with the formalism of the model checking tool. The desired property or requirement to be verified is also to be formulated in terms of state sets in the language of the model checking tool.

Re-discretization demands can arise if a system does not meet the decidability requirement. In this case partitioning of the state space needs to be continued by further abstraction of the model. The decidability requirement is met, if for a given model of the system, there exists a computational or algorithmic approach, that can determine whether the system satisfies the desired property [16] in a finite number of steps with an a priori known bound.

III. CREATING A DISCRETE EVENT MODEL VIA DISCRETE ABSTRACTIONS

The Autosub AUV [21] has adopted a modular, distributed and networked control architecture for system implementation. Subsystem nodes are distributed throughout the vehicle and carry out tasks such as guidance/mission control, control of position, depth and forward speed, navigation, actuator control, battery/power system monitoring and communication.

Discretization of the Autosub system is achieved by means of compositional abstraction. A finite state transition system illustrates the functionality of each network control node. The overall abstraction of the Autosub hybrid system model results from the parallel composition of the individual finite state transition systems. Modelling also includes material and structural (static) properties of engineering subsystems selected, hence the approach is broader than formal checking of the control systems or the multi-agent software on board.

A. Labeled transition systems (LTS)

The finite abstraction of a hybrid system generates a labeled transition system. The labeled transition system [22],[23] is a tuple

$$T_G = (Q, \Sigma, \rightarrow, Q_O, \mathcal{L}, L, Q_F) \tag{2}$$

where,

Q is the finite set of states;
Σ is the alphabet of event;
$\rightarrow \subseteq Q \times \Sigma \times Q$ is the set of transition relations;
$Q_O \subseteq Q$ is a set of initial states;
$\mathcal{L}$ is a countable set of labels;
$L: Q \rightarrow \mathcal{L}$ is the labeling function;
$Q_F \subseteq Q$ is a set of final states;

A transition $(q_1, \sigma, q_2) \subseteq Q \times \Sigma \times Q$ is denoted as $q_1 \overset{\sigma}{\rightarrow} q_2$, where q_1 is the predecessor of state q_2 and q_2 is a successor of q_1 at the occurrence of event σ.

Taking into account a formulation in terms of a multi-agent system, the transition system $T_G = (Q, \Sigma, \rightarrow, Q_O, \mathcal{L}, L, Q_F)$ is decomposed into parallel constituents, such that $T_G = T_{Agt_1} \parallel \cdots \parallel T_{Agt_n}, n \in \mathbb{N}$. The components of the transition system T_G are the transition systems with tuple:

$$T_{Agt_i} = \left(Q_{Agt_i}, \Sigma_{Agt_i}, \rightarrow_{Agt_i}, Q_{O_{Agt_i}}, \mathcal{L}_{Agt_i}, L_{Agt_i}, Q_{F_{Agt_i}}\right), i = 1, \dots, n. \tag{3}$$

A state q of the T_G transition system is denoted by $q = (q_1, \dots, q_n) \in Q$, where $Q = Q_{Agt_1} \times \dots \times Q_{Agt_n}$.

State q of the overall transition system T_G is a global state of the multi-agent system with all $\{Agt_1, \dots, Agt_n\}$ agents active at any time.

B. Natural language programming using System English (sEnglish)

Natural language programming (NLP) in sEnglish [20] is used for abstraction of continuous states to discrete states, that allows an intellectual problem formulation by the use of natural sentences, not constraining to the required technicalities of a modelling input language. NLP in sEnglish is theoretically well founded method to record relationships of models and procedures on board the AUV and in its environment. A major advantage of using the NLP paradigm is that it opens up the potentials of the human brain's natural abstraction ability of our environment. Any engineer can benefit from doing programming using this development paradigm where conceptual structures are defined, then top level sentences are explained by other sentences and in return those by other sentences, until there is no more abstracted detail needed and plain computer code is the meaning of the lower level sentences. An NLP document also looks like a technical paper or report and can easily be understood and shared in team work. In our approach NLP is used to formulate functionality of the AUV at various abstraction levels and to link the abstractions to Stateflow™ charts and to MCMAS that is a model checker for multi-agent systems [24, 25]. Further details of NLP are illustrated in Section VI.

The ontology described in (4-8) has been chosen to provide the conceptual representation of the labeled transition system that is resulted by the natural language formulation performed discrete abstraction process.

MAS (4) illustrates the concept of the multi-agent system formed by its consituents, i.e. of the set of agents;

$$MAS = \bigcup Agt_i, i \in \mathbb{N} \tag{4}$$

The concept of an agent Agt_i is defined by a tuple (5) consisting of its name, its set of operational modes, its set of event attributes and the set of possible transition relations.

$$Agt_i = (agent_name, \bigcup operational_mode_{ij}, \bigcup event_attribute_{ik}, \bigcup transition_relation_{il}),\ i, j, k, l \in \mathbb{N} \tag{5}$$

The operational mode concept (6) refers to the local states Q_{Agt_i} that an agent can find itself in during its operation. The set of operational modes of an agent can be defined by the name of its finite local states Q_{Agt_i}; or in terms of local variables, whose set of ordered pairs can ultimately define the agent's operational modes.

$$operational_mode_{ij} = (local_state_name, variable_definition, variable_name, variable_type, variable_valuation, value_range_minimum, value_range_maximum)\ i,j \in \mathbb{N} \quad (6)$$

The event attribute concept (7) grasps the relationship between the LTS set of labels $\ell_{ik} \subseteq \mathcal{L}_{Agt_i}$ (actions) that when assigned to a local state $q_{ij} \in Q_{Agt_i}$ by the labeling function, results a consequent alphabet of events $\sigma_{ik} \subseteq \Sigma_{Agt_i}$. Hence, the function $h: Q_{Agt_i} \times \mathcal{L}_{\mathrm{Agt_i}} \to \Sigma_{Agt_i}$ defines the mapping from the Cartesian product of the finite set of local states Q_{Agt_i} and the agent's countable set of labels $\mathcal{L}_{Agt_i}$, to the alphabet of events Σ_{Agt_i}.

The labeling function $L_{Agt_i}: Q_{Agt_i} \to \mathcal{L}_{Agt_i}$ assigns the actions to each operational mode, while the function $h: Q_{Agt_i} \times \mathcal{L}_{Agt_i} \to \Sigma_{Agt_i}$ assigns the triggered events to the actions performed in a given operational mode, so that $\sigma_{ik} = f(q_{ij}, \ell_{ik})$.

$$event_attribute_{ik} = (event_name, action_label, host_local_state)\ , \quad i,k \in \mathbb{N} \quad (7)$$

Each transition relation (8) shows the possible evolution of the agent from one local state to another.

The operational mode evolution concept consists of the set of all transitions of an agent $\to_{Agt_i} \subseteq Q_{Agt_i} \times \Sigma_{Agt_i} \times Q_{Agt_i}$, that is the set of ordered pairs $(q_{ij}, \sigma_{ik}, q_{ij'}) \subseteq Q_{Agt_i} \times \Sigma_{Agt_i} \times Q_{Agt_i}$, $i,j,j',k \in \mathbb{N}$ where q_{ij} is the predecesor state of $q_{ij'}$, and $q_{ij'}$ is the successor state of q_{ij}, at the occurrence of event σ_{ik}. The components of transition relation's tuple are therefore, the q_{ij} source local state from where the agent evolves into the $q_{ij'}$ destination local state, at the occurrence of event σ_{ik}, the latter being triggered when the agent Agt_i performs the action $\ell_{ik} \in \mathcal{L}_{Agt_i}$ assigned to its source local state, such that $\ell_{ik} = L_{Agt_i}(q_{ij})$.

$$transition_relation_{ik} = (source_local_state, destination_local_state, event_name, action_label, action_performing_agent), i,k \in \mathbb{N} \quad (8)$$

C. Stateflow formalism

The Simulink™ and Stateflow™ software tools developed by Mathworks became an industry standard and are a popular choice for modelling hybrid systems and embedded applications. Stateflow charts allow the creation of discrete state transition systems based on hierarchical state machines. Simulink blocks provide the ability to model the continuous dynamics corresponding to the discrete states.

The discrete model obtained by compositional abstraction and modelled by the use of natural language sentences, translates into the Stateflow™ formalism for the purpose of simulation of the discrete event system and automatic code generation. The natural language model uses a MOL (machine ontology language) ontology and states operational mode attributes and transitions in terms of human concepts in sentences while permitting unambiguous translation into a formal agent definition.

Adopting the notation from [26], a Stateflow chart is described by the tuple

$$SF = (D, E, S, Fct, T, f) \quad (9)$$

where,

$D = D_I \cup D_O \cup D_L$ is a finite set of typed variables representing data objects in the Stateflow chart; D is partitioned into input variables D_I, output variables D_O and local variables D_L;

$E = E_I \cup E_O \cup E_L$ is a finite set of events that is partitioned into input variables E_I, output variables E_O and local variables E_L;

S is a finite set of states, where each state is a tuple:

$$S = (entry\ action, exit\ action, during\ action, on\ event\ action) \quad (10)$$

An action can contain a function call, can be an assignment of an expression to a variable or can be an event broadcast.

Fct is a finite set of Matlab™ or graphical functions that can be included as a call statement in the expression of a state's *entry action*, *exit action*, *during action* or *on event action*;

T is a finite set of transitions, where each transition is a tuple $T = (src, dst, e, c, ca, ta)$. $src \in S$ represents the source state, $dst \in S$ is the destination state, $e \in E$ is an event, c is the condition expression, ca is the condition action and ta is the transition action;

$f: S \to (\{and, or\} \times 2^S)$ is a mapping from the finite set of states to the Cartesian product of the states' decomposition type and the power set of S. Parallel (and) state decomposition represents states at the appropriate level in the hierarchy that are always active at the same time. Exclusive (or) decomposition describes states that are mutually exclusive.

$C \in 2^S \times D$ represents the configuration of a Stateflow chart and is a tuple consisting of the set of active states and a valuation for all the variables from the finite set D. D denotes the set of all valuations of a variable set D.

D. MCMAS

MCMAS is a model checker for multi-agent systems, intended for the automatic verification of formulae with temporal and epistemic properties in deontic interpreted systems (DIS).

MCMAS uses an extended variant of an interpreted system's formalism [27]. Each agent from a system of agents can be modelled by means of a set of local states, a set of actions, a protocol and an evolution function [24, 25]. An agent ($i \in \{1, \dots, n\}, n \in \mathbb{N}$) is characterized by a finite set of local states

L_i, and by a finite set of actions Act_i. The protocol P_i is a rule that establishes which action can be performed by an agent in a given local state. The protocol function $P_i : L_i \rightarrow 2^{Act_i}$ assigns a set of actions to a local state. The evolution function $t_i: L_i \times L_E \times Act_1 \times \dots Act_n \times Act_E \rightarrow L_i$ determines how the local state of an agent evolves based on the agent's local state, on the local state of a special agent E modelling the environment, and on the actions of all agents.

As addressed in [24] and [25], an element $g \in S$ of the Cartesian product of the agents' local states $S = L_1 \times \dots \times L_n \times L_E$ is called a global state. An element $\alpha \in Act$ of the Cartesian product of the agents' action $Act = Act_1 \times \dots \times Act_n \times Act_E$ is a tuple of actions and is referred to as a joint action. The evolution of the global states of the system is described by the function $t: S \times Act \rightarrow S$. An element $g \in S$ represents a global state. Hence, given a global state g, the local state of agent i in the global state g is denoted by the symbol $l_i(g)$.

Given a set $I \subseteq S$ of possible initial global states, the protocols and the evolution functions generate a set $G \subseteq S$ of reachable global states, obtained by all the possible runs of the system. Given a set of atomic propositions P, the evaluation relation $V \subseteq S \times P$ completes the description of the interpreted system.

Interpreted systems were extended to include the notion of correct behaviour [28]. According to this extension, the set of local states L_i is partitioned into the non-empty set G_i of allowed or correct states, and a set R_i of disallowed states, such that $L_i = G_i \cup R_i$ and $G_i \cap R_i = \emptyset$.

Hence, given a set of agents $\{1, \dots, n\}$, a deontic interpreted system (DIS) is defined as the tuple:

$$DIS = \langle (G_i, R_i, Act_i, P_i, t_i)_{i \in \{i, \dots, n\}}, (G_E, R_E, Act_E, P_E, t_E), I, V \rangle \tag{11}$$

The following language applies to a deontic interpreted system:

$$\varphi ::= p | \neg\varphi | \\ | \varphi \vee \varphi | EX\varphi | EG\varphi | E[\varphi U \psi] | K_i\varphi | E_\Gamma\varphi | C_\Gamma\varphi | D_\Gamma\varphi | O_i\varphi | \widehat{K}_i^\Gamma \varphi \tag{12}$$

With the language above, the interpreted systems provides semantics to reason about temporal and epistemic properties [24, 25]. In this grammar $p \in P$ is an atomic proposition, EX EG and EU are standard computation tree logic (CTL) operators [12]. The remaining CTL operators can be derived from the above. The formula $K_i\varphi$ ($i \in \{1, \dots, n\}$) expresses "agent i knows φ". The symbol Γ denotes a group of agents. The formula $E_\Gamma\varphi$ expresses "everybody in group Γ knows φ"; the formula $C_\Gamma\varphi$ expresses "φ is common knowledge in group Γ"; the formula $D_\Gamma\varphi$ expresses "φ is distributed knowledge in group Γ"; the formula $O_i\varphi$ expresses that "under all the correct alternatives for agent i, φ holds"; lastly, the formula $\widehat{K}_i^\Gamma\varphi$ expresses the knowledge that agent i has on the assumption that all agents in group Γ are functioning correctly.

MCMAS inherently handles the interpretation of formulae by associating a Kripke model to the deontic interpreted systems. For further details on the Kripke model representation and the inductive definition of the satisfaction conditions, the user is asked to refer to [24] and [25].

IV. Translation Tool from Natural Language Programming Ontology to Discrete Event System Stateflow Formalism

An NLP-based natural language agent model uses a MOL (machine ontology language) ontology and states operational mode attributes and transitions in terms of human concepts in sentences while permitting unambiguous translation into a formal agent definition. We have developed a tool to translate the natural language programming abstractions of agent operations to the discrete event system representation in Stafelow formalism. This translation process has been intended for the purpose to execute the simulations of the discrete event system in the Stateflow environment, and in addition to provide means of automatic code generation and deployment to embedded targets as stand-alone applications. A formal definition of the multi-agent system resulting from this translation process can be described as follows.

Given a multi-agent system MAS of n agents in natural language statements, the mapping of translation is denoted by $f(MAS) = (and, \{Agt_1, \dots, Agt_n\})$, where MAS is the Stateflow chart with parallel state decomposition. Due to this representation the states $\{Agt_1, \dots, Agt_n\}$ from this level of the hierarchy are active at the same time, hence the agents can evolve simultaneously.

An agent Agt_i ($i \in \{1, \dots, n\}, n \in \mathbb{N}$) is described by the mapping function $f(Agt_i) = (or, \{s_{i1}, \dots, s_{im}\})$, where $s_{ij} \in S_i$, $j \in \{1, \dots, m\}, m \in \mathbb{N}$) are local states of the respective agent. In this implementation the agent Agt_i is a superstate with exclusive state decomposition, whose substates are the agent's local states. Superstate Agt_i being of exclusive decomposition, only one of its local state s_{ij} can be active at a time.

The finite set of actions for agent Agt_i is realized and defined as a group of functions Fct_i that reside in and are declared local to the Agt_i superstate (Fig. 3).

The actions that can be performed by an agent Agt_i in a given local state s_{ij} is denoted by $fct_{ik} \subseteq Fct_i$, $k \in \{1, \dots, p\}, p \in \mathbb{N}$). When an action is performed within the local state, a transition with a triggered event label can be activated. If the source state of the transition is active, the transition is taken, thus enabling the agent to evolve from one local state to another. Hence, an event label can activate a transition using an underlying event broadcast mechanism that occures in a local state's action.

The local state of agent Agt_i in an initial global state is marked by assigning a s_{ij} local state as default.

A MOL ontology has been defined for the conceptual representation of the Stateflow objects in a NLP along with sentence definitions, to endow the creation of Stateflow charts with the use of sEnglish natural language sentences. This

enables direct linking of human concepts of operation to the formal definition of agents.

The ontology used for the conceptual representation of the labeled transition system provide the type of objects, their properties and relations, that are processed and transposed into the Stateflow ontology representation to create the discrete event system representation as Stateflow charts.

V. TRANSLATION TOOL FROM NATURAL LANGUAGE PROGRAMMING ONTOLOGY TO MULTI-AGENT MODEL CHECKING

The natural language programming abstraction of the multi-agent systems is also linked to the interpreted script of the multi-agent model checker MCMAS for verification.

An ontology defined for the ISPL representation encapsulates the representational requirements of the multi-agent model checker's input language.

The agent constituents of the multi-agent system are modelled by means of a set of local states definable by state names or in terms of local variables, a set of actions, a protocol and an evolution function.

In terms of the labeled transition system representation, the DIS (deontic interpreted system) action Act_i definition of an agent i $(i \in \{1, \ldots, n\}, n \in \mathbb{N})$ relate to the countable set of labels $\mathcal{L}_{Agt_i}$ of an agent Agt_i from the LTS representation.

The DIS protocol function represents the LTS labeling function assigning a subset of labels to a finite set of states, i.e. the actions that the agent can perform in a given local state. The protocol function for a given local state $q_{ij} \in Q_{Agt_i}$ is expressed as the subset $\ell_{ik} \subseteq \mathcal{L}_{Agt_i}$, where $\ell_{ik} = L_{Agt_i}(q_{ij})$, are the set of labels assigned to the local state $q_{ij} \in Q_{Agt_i}$ $(i \in \{1, \ldots, n\},\ j \in \{1, \ldots, m\},\ k \in \{1, \ldots, o\}, n, m, o \in \mathbb{N})$.

The DIS evolution function implements the mapping to a given local state, regarded as the succesor state, from all its possible predecessor states. The evolution function of a given local state $q_{ij'} \in Q_{Agt_i}$ is expressed as the set of all ordered pairs of actions and local states $(\ell_{ik}, q_{ij}) \subseteq \mathcal{L}_{Agt_i} \times Q_{Agt_i}$, such that all transitions $(q_{ij}, \sigma_{ik}, q_{ij'})$ occurred at the event $\sigma_{ik} = h(q_{ij}, \ell_{ik})$ have the same successor state $q_{ij'}$, $(i \in \{1, \ldots, n\},\ j \in \{1, \ldots, m\},\ k \in \{1, \ldots, o\}, m, n, o \in \mathbb{N})$.

Prior to verification, the description of the multi-agent system is completed with the definition of the evaluation function to establish the atomic propositions, the initial states, agent groups, fairness statements (as defined in ISPL) and formulae to be verified.

VI. MODELLING EXAMPLE

To illustrate the aforementioned problem formulation using natural language programming and its translation to the model checkers' input language and to Stateflow, an example of the Collision and Obstacle Avoidance process/agent (see Fig. 3) is presented [21],[29].

Sensor measurements from acoustic doppler current profilers (ADCP) and from a forward looking echosounder provide environmental awareness capability for the collision and obstacle avoidance process. In the following illustration, the 'limited headroom acknowledged' and 'obstacle ahead acknowledged' are events that are triggered by the 'limited headroom event received from adcp', respectively the 'obstacle ahead event received from echosounder' actions performed by the 'obstacle avoidance' agent. The 'limited headroom event received from adcp' action illustrate that both the vehicle's altitude above the seafloor and range to the ice overhead crosses a permitted threshold value. The 'obstacle ahead event received from echosounder' activity indicates that there is an obstruction on the vehicle's path.

'Waiting' is the active state during normal operation of the AUV, when the collision and obstacle avoidance is not taking any action [29]. When triggered by the 'limited headroom acknowledged' or 'obstacle ahead acknowledged' event, the state of the avoidance process switches to 'retreating'. The actions that can be taken in this state consist of a 180 degree turn and a retreat along the same path it had come along, to a preset safe retreat distance. When the retreat distance is covered, the 'try new track' state is entered, during which the vehicle will try a new track, parallel to the original. If the vehicle encounters an obstacle, it will switch back to 'retreating' state. If the vehicle has covered a clearance distance on the alternative path, the obstacle avoidance process is finished, the 'trying new track' state is exited and the 'waiting' state is entered.

The natural language programming paradigm uses sentences, sections, subsections, ontology description sections, topic classification, problem area description, etc. An ontology of concepts is developed, defining for each concept an associated object class with the most relevant attributes that are needed in the problem area addressed by the natural language program.

The following ontology is used to illustrate the concepts of the LTS discrete model resulted by the natural language sentence formulation:

```
>multi-agent system
   @constituents: set of agents

>agent
   @name: char
   @operational profile: set of operational modes
   @action triggered events: set of event attributes
   @operational mode evolution: set of transition relations

>operational mode
   @local state name: cell
   @variable definition: cell
   @variable name: cell
   @variable type: cell
   @variable valuation: cell
   @value range minimum: cell
   @value range maximum: cell
```

>event attribute
@event name: cell
@action label: cell
@host local state: cell

>transition relation
@source local state: cell
@destination local state: cell
@event name: cell
@action label: cell
@action performing agent: cell

>ispl representation
@ispl agents: set of interpreted systems agents
@ispl evaluation function: char
@ispl initial states: char
@ispl groups: char
@ispl fairness formulae: char
@ispl verification formulae: char

>interpreted systems agent
@agent name: char
@observable variables: char
@variables: char
@red states: char
@actions: char
@protocol function: char
@evolution function: char

The 'multi-agent system', 'agent', 'operational mode', 'event attributes' and 'transition relation' object classes relate to the discrete model of a generic agent definition formulated in NLP (formulated in "system English" or for short in "sEnglish"), while the 'ispl representation' and 'interpreted systems agent' object classes are utilized for the representation of the agent in the model checker's input language.

Natural language programming advocates the top-down method for problem formulation. The top level sentence creates the shell of the generic agent and is defined by a sequence of other sentences that use concepts from the ontology. Every sentence that has been used is defined in terms of other sentences, or by a simple sentence that executes code written in an associated high level programming language.

In the following passage an example formulation of the obstacle avoidance process is presented, using sEnglish.

Define agent obstacle avoidance.
Define the agent name as 'obstacle avoidance'. Define all operational modes of obstacle avoidance. Define all actions of obstacle avoidance. Define all events generated by the obstacle avoidance.

Define all operational modes of obstacle avoidance.
The 'waiting' is an operational mode of 'obstacle avoidance'. The preconditions of 'waiting' of 'obstacle avoidance' can be 'clearance distance covered by vehicle, maintain waiting'. The exit conditions of 'waiting' of 'obstacle avoidance' can be 'limited headroom acknowledged while waiting, obstacle ahead acknowledged while waiting, maintain waiting'.
The 'retreating' is an operational mode of 'obstacle avoidance'. The preconditions of 'retreating' of 'obstacle avoidance' can be 'limited headroom acknowledged while trying new track, obstacle ahead acknowledged while trying new track, limited headroom acknowledged while waiting, obstacle ahead acknowledged while waiting, maintain retreating'. The exit conditions of 'retreating' of 'obstacle avoidance' can be 'retreat distance covered by vehicle, limited headroom acknowledged while retreating, obstacle ahead acknowledged while retreating, maintain retreating'.
The 'trying new track' is an operational mode of 'obstacle avoidance'. The preconditions of 'trying new track' of 'obstacle avoidance' can be 'retreat distance covered by vehicle, limited headroom acknowledged while retreating, obstacle ahead acknowledged while retreating, maintain trying new track'. The exit conditions of 'trying new track' of 'obstacle avoidance' can be 'clearance distance covered by vehicle, limited headroom acknowledged while trying new track, obstacle ahead acknowledged while trying new track, maintain trying new track'.

Define all actions of obstacle avoidance.
The 'turn and retreat, preset safe retreat distance from obstacle was reached, obstacle ahead event received from echosounder, limited headroom event received from adcp' actions can be performed by 'obstacle avoidance' in 'retreating' host operational mode.
The 'try new track, vehicle completed twice the retreat distance and attempts to reacquire original track, obstacle ahead event received from echosounder, limited headroom event received from adcp' actions can be performed by 'obstacle avoidance' in 'trying new track' host operational mode.
The 'avoidance process is idling, obstacle ahead event received from echosounder, limited headroom event received from adcp' actions can be perfomed by 'obstacle avoidance' in 'waiting' host operational mode.

Define all events generated by the obstacle avoidance.
The 'clearance distance covered by vehicle' is an event of 'obstacle avoidance' triggered by the action 'vehicle completed twice the retreat distance and attempts to reacquire original track'.
The 'maintain trying new track' is an event of 'obstacle avoidance' triggered by the action 'try new track'. The 'maintain retreating' is an event of 'obstacle avoidance' triggered by the action 'turn and retreat'. The 'maintain waiting' is an event of 'obstacle avoidance' triggered by the action 'avoidance process is idling'.
The 'retreat distance covered by vehicle' is an event of 'obstacle avoidance' triggered by the action 'preset safe retreat distance from obstacle was reached'.
The 'obstacle ahead acknowledged while waiting, obstacle ahead acknowledged while retreating, obstacle ahead acknowledged while trying new track' is an event of 'obstacle avoidance' triggered by the action 'obstacle ahead event received from echosounder'.
The 'limited headroom acknowledged while waiting, limited headroom acknowledged while retreating, limited headroom acknowledged while trying new track' is an event of 'obstacle avoidance' triggered by the action 'limited headroom event received from adcp'.

```
Agent Obstacle_avoidance
    Vars:
        state : {waiting, retreating, trying_new_track};
    end Vars
    RedStates:
    end RedStates
    Actions = { avoidance_process_is_idling, limited_headroom_event_received_from_adcp, obstacle_ahead_event_received_from_echosounder,
preset_safe_retreat_distance_from_obstacle_was_reached, try_new_track, turn_and_retreat,
vehicle_completed_twice_the_retreat_distance_and_attempts_to_reacquire_original_track };
    Protocol:
        state = retreating : {limited_headroom_event_received_from_adcp, obstacle_ahead_event_received_from_echosounder,
preset_safe_retreat_distance_from_obstacle_was_reached, turn_and_retreat};
        state = trying_new_track : {limited_headroom_event_received_from_adcp, obstacle_ahead_event_received_from_echosounder,
try_new_track, vehicle_completed_twice_the_retreat_distance_and_attempts_to_reacquire_original_track};
        state = waiting : {avoidance_process_is_idling, limited_headroom_event_received_from_adcp,
obstacle_ahead_event_received_from_echosounder};
    end Protocol
    Evolution:
        state = retreating if (((obstacle_avoidance.Action = limited_headroom_event_received_from_adcp) or (obstacle_avoidance.Action =
obstacle_ahead_event_received_from_echosounder)) and ((state = trying_new_track) or (state = waiting))) or ((obstacle_avoidance.Action = turn_and_retreat)
and (state = retreating));
        state = trying_new_track if (((obstacle_avoidance.Action = limited_headroom_event_received_from_adcp) or (obstacle_avoidance.Action
= obstacle_ahead_event_received_from_echosounder) or (obstacle_avoidance.Action = preset_safe_retreat_distance_from_obstacle_was_reached)) and (state =
retreating)) or ((obstacle_avoidance.Action = try_new_track) and (state = trying_new_track));
        state = waiting if ((obstacle_avoidance.Action = vehicle_completed_twice_the_retreat_distance_and_attempts_to_reacquire_original_track)
and (state = trying_new_track)) or ((obstacle_avoidance.Action = avoidance_process_is_idling) and (state = waiting));
    end Evolution
end Agent
```

Table 1. Obstacle avoidance agent definition in ISPL formalism

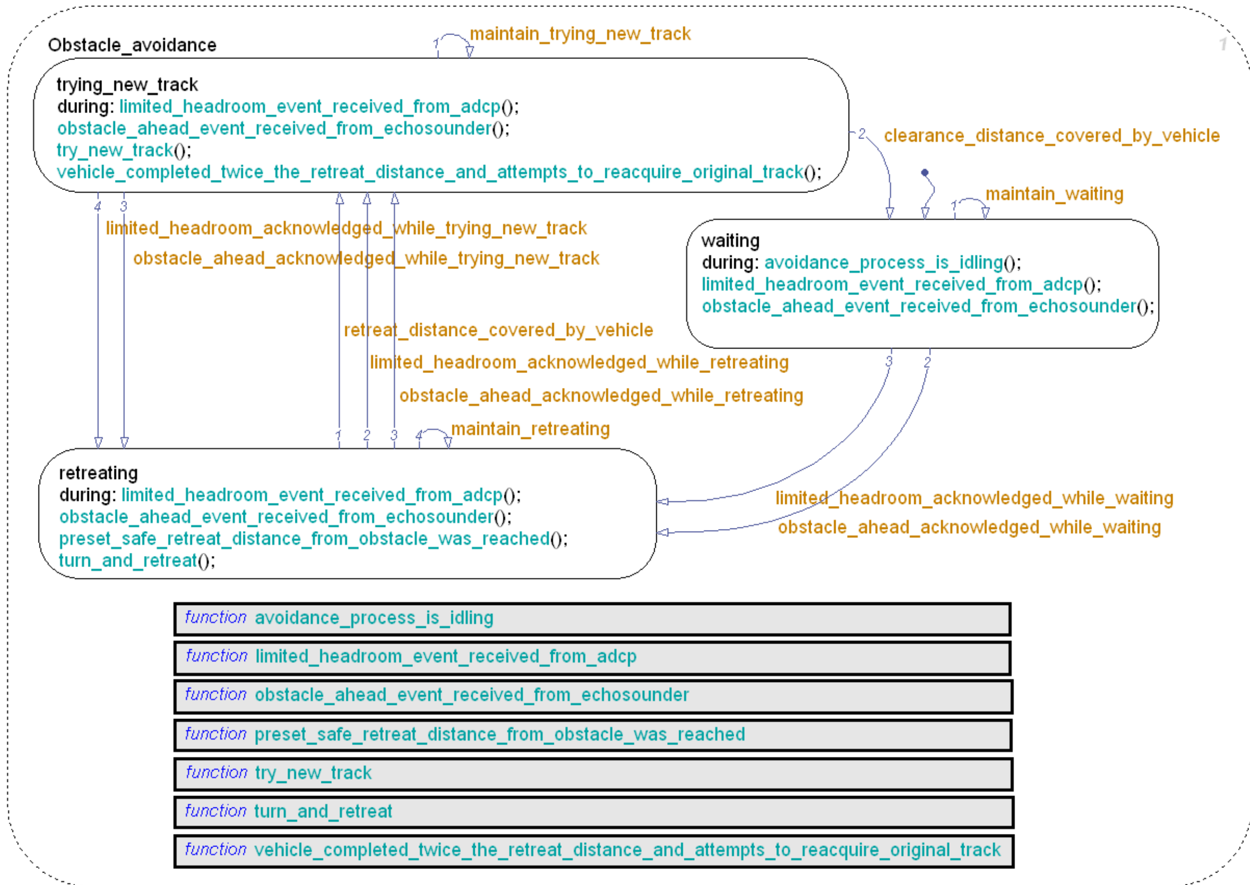

Fig. 3. Obstacle avoidance Stateflow chart representation

Table 1 illustrates the agent definition framework in ISPL formalism, resulted by the translation process from the natural language formulation and ontology representation. The agent 'Obstacle_avoidance' is modelled by means of a set of local states, i.e. 'waiting', 'retreating', 'try_new_track'; the set of actions that the agent can perform; the protocol function illustrating the actions that the agent can perform in a specific local state; and lastly by the evolution function, that shows how the local state of the agent evolves based on its own and the remaining agent's local states and the actions that are performed there.

Fig. 3 details the discrete event system representation of the agent using the Stateflow formalism.

Elaborate specifications can be formulated with sEnglish in order to provide the verification requirements that the multi-agent system representation of the Autosub AUV must satisfy.

> *Add the definitions of the verification formulae to the multi-agent system description.*
>
> The sentence 'it is always true that, if, Obstacle avoided acknowledgement is received, then, the, Mission controller, knows that, the, Position controller, knows that, Obstacle is avoided' represents a requirement that needs to be verified.
>
> The sentence 'it is always true that, it will not hold that, it is possible to get to a state where, Launch procedure starts, and, Vehicle ready, does not hold' represents a requirement that needs to be verified.

At the execution of the above sEnglish program the following logic formulae are resulted (13,14):

$$AG\left(Obstacle_avoided_acknowledgement_is_received \rightarrow K\big(Mission_controller, K(Position_controller, Obstacle_is_avoided)\big)\right) \quad (13)$$

$$AG\big(!\,EF(Launch_procedure_start \text{ and } !\,Vehicle_ready)\big) \quad (14)$$

where,

$Obstacle_avoided_acknowledgement_is_received$, $Vehicle_ready$, $Launch_procedure_start$ are atomic propositions;

$Mission_controller$, $Position_controller$ are ISPL agents;

A is the "for all computation paths" CTL path quantifier;

G is the "always" or "globally" temporal operator specifying that a property hold at every state on the path;

K is an epistemic operator, expressing the knowledge of an agent;

E is the "for some computation paths" CTL path quantifier;

F is the "eventually" or "at some point in the future" temporal operator specifying that a property will hold at some state on the path;

$\rightarrow$ is the implication logical connective;

and is the conjunction logical connective;

! is the negation logical connective.

VII. CONCLUSIONS

This paper has introduced a complete methodology for the integrity and fault assessment system of complex autonomous engineering systems such as an autonomous underwater vehicle, by formal verification. Modelling can also include material and structural (static) properties of engineering subsystems selected, hence the approach is broader than formal checking of the control systems. The first system models are obtained by hybrid system modelling. Then natural language models are abstracted from the hybrid models in NLP. This NLP-based model can be automatically compiled into Stateflow and ISPL for simulation or formal verification by model checking that can test whether safety critical temporal logic statements hold. A crucial step of our procedure is the bisimulation based abstraction in terms of NLP statements that not only help verification but can be a vital part of the agents being able to report problems to human operators in English.

Component failures, that may affect or endanger mission success, are represented as discrete events in the MCMAS system. Reliability can be tested under various assumptions of component reliability and within the given limits of accuracy of the discrete abstractions used for the environment and internal feedback loops.

The multi-agent IFAS verification system created by us also allows the inclusion of all the network communications and behavior of human controllers.

Further work will be concerned with extending the IFAS system to an iterative design process of autonomous vehicles. Also the human operator's role as an agent will be a focus of our interest. Eliminating human errors by warning and safety systems is vitally important to mission success and to avoid loss of vehicles.

REFERENCES

1. Chance, T.S., *AUV surveys - extending our reach, 24000 km later*, in *Proceedings of 13th Unmanned Untethered SubmersibleTechnology 2003, AUSI*. 2003: *New Hampshire*.
2. Griffiths, G., et al., *On the reliability of the Autosub autonomous underwater vehicle. Underwater Technology*, 2003. **25**(4): p. 175 - 184.
3. Podder, K., et al., *Reliability growth of an autonomous underwater vehicle - Dorado*, in *Proc. Oceans'04, Kobe, Japan. MTS/IEEE*. 2004. p. 856-862.
4. Strut, J., *Report of the inquiry into the loss of Autosub II under the Fimbulisen*, *in on line, UK, available at url:* http://eprints.soton.ac.uk/41098/01/AUV_Loss_Report_NOCS_R%26C_12.pdf. 2006, *National Oceanography Centre*: *Southampton*.
5. Griffiths, G., *The technology and applications of autonomous underwater vehicles*. 2003, London: Taylor and Francis.
6. Hamilton, K., *Practical self-diagnosing AUVs*, in *Proceedings UUVS'02*. 2002. p. 1-5.
7. Washington, R., et al., *Autonomous rovers for Mars exploration*, in *Proc. of the 1999 IEEE Aerospace Conference*. 1999.
8. Williams, B.C. and P.P. Nayak, *A model-based approach to reactive self-configuring systems*, in *Workshop on Logic-Based Artificial Intelligence*. 1999: *Washington, DC*.
9. O´Connor, M., et al., *A bottom-up approach to verification of hybrid model-based hierarchical controllers with application to underwater vehicles*, in *14th International Symposium on Unmanned Untethered Submersible Technology, UUST05*. 2005. p. 1-8.
10. Davis, D.T., *Automated parsing and conversion of vehicle-specific data into autonomous vehicle control language (AVCL) using context free*

grammars and xml data binding, in *14th International Symposium on Unmanned Untethered Submersible Technology, UUST05*. 2005.
11. Smith, B., et al., *Validation and verification of the remote agent for spacecraft autonomy*, in *Proceedings of the IEEE Aerospace Conference, Snowmass*. 1999: *CO, USA*.
12. Edmund M. Clarke, J., O. Grumberg, and D.A. Peled, *Model Checking*. 1999, London, England: The MIT Press.
13. Lussier, B., et al., *On fault tolerance and robustness in autonomous systems*, in *Proceedings of the 3rd IARP-IEEE/RAS-EURON Joint Workshop on Technical Challenges for Dependable Robots in Human Environments*. 2004: Manchester, UK.
14. Muscettola, N., et al., *Idea: Planning at the core of autonomous reactive agents*, in *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*. 2002.
15. Varaiya, P., *Design, Simulation, and Implementation of Hybrid Systems*, in *Applications and Theory of Petri Nets 1999: 20th International Conference, ICATPN'99, Williamsburg, Virginia, USA, June 1999. Proceedings*. 1999. p. 71-71.
16. Alur, R., et al., *Discrete abstractions of hybrid systems.* Proceedings of the IEEE, 2000. **88**(7): p. 971-984.
17. Davoren, J.M. and A. Nerode, *Logics for hybrid systems.* Proceedings of the IEEE, 2000. **88**(7): p. 985-1010.
18. Carloni, L.P., et al., *Languages and Tools for Hybrid Systems Design*. 2006: Foundations and Trends in Electronic Design Automation, Vol. 1, No 1/2 (2006) 1–193, Now.
19. Lygeros, J., C. Tomlin, and S. Sastry, *The Art of Hybrid Systems*. 2001.
20. Veres, S.M., *Natural language programming of agents and robotic devices*. 2008, London: SysBrain Ltd.
21. McPhail, S.D. and M. Pebody, *Navigation and Control of an Autonomous Underwater Vehicle Using a Distributed, Networked, Control Architecture* Underwater Technology: The International Journal of the Society for Underwater, 1998. **23**(12): p. 19-30.
22. Chutinan, A., A. Chutinan, and B.H. Krogh. *Approximating quotient transition systems for hybrid systems*. in *American Control Conference, 2000. Proceedings of the 2000*. 2000.
23. Lafferriere, G., G. Pappas, and S. Yovine, *A New Class of Decidable Hybrid Systems*, in *Hybrid Systems: Computation and Control*. 1999. p. 137-151.
24. Raimondi, F. and A. Lomuscio, *Automatic verification of deontic interpreted systems by model checking via OBDDs.* Journal of Applied Logic. Special issue on Logic-based agent verification, 2005. **5**(2): p. 235-251.
25. Raimondi, F., *Model Checking Multi-Agent Systems*, in *Department of Computer Science*. 2006, University College London: London. p. 142.
26. Tiwari, A., *Formal semantics and analysis methods for Simulink Stateflow models, Technical report*. 2002, SRI International.
27. Fagin, R., *Reasoning about knowledge* 1995, Cambridge: MIT Press. 477.
28. Lomuscio, A. and M. Sergot, *Deontic Interpreted Systems.* Studia Logica (Special Issue on The Dynamics of Knowledge), 2003. **75**(1): p. 63-92.
29. Pebody, M., *Autonomous underwater vehicle collision avoidance for under-ice exploration* in *Proceedings of the I MECH E Part M*. 2008, Professional Engineering Publishing. p. 53-66.